20. Text Processing and Boolean Models

INFO 202 - 5 November 2013

Bob Glushko

Plan for Today's Class

Models for Information Retrieval

Text Processing Operations and Challenges

The Boolean Model

Models of Information Retrieval [1]

The core problems of information retrieval are finding relevant documents and ordering the found documents according to relevance

The IR model explains how these problems are solved:

- ...By specifying the representations of queries and documents in the collection being searched
- ...And the information used, and the calculations performed, that order the retrieved documents by relevance
- (And optionally, the model provides mechanisms for using relevance feedback to improve precision and results ordering)

Models of Information Retrieval [2]

BOOLEAN model -- representations are sets of index terms, set theory operations with Boolean algebra calculate relevance as binary

VECTOR models -- representations are vectors with non-binary weighted index terms, linear algebra operations yield continuous measure of relevance

Models of Information Retrieval [3]

STRUCTURE models -- combine representations of terms with information about structures within documents (i.e., hierarchical organization) and between documents (i.e. hypertext links and other explicit relationships) to determine which parts of documents and which documents are most important and relevant

PROBABILISTIC models -- documents are represented by index terms, and the key assumption is that the terms are distributed differently in relevant and non relevant documents.

What is a "Document" in Information Retrieval?

A document is any individually retrievable item in the "pile of text" that makes up the COLLECTION

Sometimes the boundaries that define documents are obvious or conventional (web search returns a web page), but sometimes they aren't

"Carving up" or "chunking" large documents into smaller text passages may be required for some collections or some user interfaces

A collection might contain any number of documents; web search engines index billions of pages

(FLASHBACK) Identifying Information Components

Any piece of information that can be addressed and manipulated by a person or process as a discrete entity

Any piece of information with a unique name or identifier

Any piece of information that is self-contained and comprehensible on its own

Text Processing: Motivation (1)

Not all words are equally useful indicators of what a document is about

Nouns and noun groups carry more "aboutness" than adjectives, adverbs, and verbs

Very frequent words that occur in all or most documents add NOISE because they cannot discriminate between documents

So it is worthwhile to pre-process the text of documents to select a smaller set of terms that better represent them; these are called the INDEX terms

Text Processing: Motivation (2)

Extracting a set of terms for describing the documents in some collection is the foundation for information retrieval

But IR locates documents for some person making a query, and getting value from those documents and satisfying the information needs involves work by that person

If the agent with the information need isn't a person, but some computational process, the document alone isn't likely to satisfy the information need

The agent will need a richer, more semantic representation of the document contents

Text Processing as a "Pipeline"

Text processing is often described as a pipeline - a sequence of processing stages

The pipeline starts with "raw" text, and each processing step "refines" the text in some way

The analogy to refining of some substance like petroleum is clear - you start with "dirty" input and progressively remove impurities, stopping when the quality is adequate for your purposes (cf. diesel fuel vs high-octane jet fuel)

Text Processing: Operational Overview (1)

DECODING -- extracting the text to be processed from its stored representation

FILTERING -- creating a stream of characters by removing formatting or non-semantic markup

TOKENIZATION -- segmenting the character stream into linguistic units

NORMALIZATION -- creating equivalence classes from tokens with superficially different character sequences

Text Processing: Operational Overview (2)

STEMMING -- removing affixes and suffixes to allow the retrieval of syntactic and morphological variations of query terms

STOPWORD ELIMINATION -- remove words that poorly discriminate between documents

SELECTING INDEX TERMS -- choosing word/stems (or groups of them) as indexing elements

CREATING AUXILIARY STRUCTURES -- like a THESAURUS

Decoding

"Raw" text is a series of digital bit patterns that encode a character set

Tokenizing raw text requires an understanding of the text encoding scheme, and there are many single- or multi-byte encoding schemes

Determining the encoding can be easy (file extensions or metadata) -- but not always

Text encoding specs are well-documented (http://www.wotsit.org /list.asp?fc=10) but "commercial products can easily live or die by the range of encodings they support"

ASCII

The ASCII scheme was standardized in the 1960s when computer memory was expensive and most computing was in English-speaking countries, so it is minimal and distinguishes only 128 characters

But ASCII is inadequate for most languages that have larger character sets with characters like È, æ, ç - nor can it handle mathematical characters like λ,σ, etc.

Wikipedia has articles in many non-ASCII character sets

Unicode

Two encodings can use the same number for two different characters, or use different numbers for the same character

The Unicode standard was designed to overcome this problem

Unicode 6.0 has room to encode 109,449 characters for all the writing systems in the world, so a single standard can represent the characters of every existing language, even "dead" ones like Sumerian and Hittite

Each character is assigned a number, called its CODE POINT

Unicode encodes the scripts used in languages, rather than languages per se, so there only needs to one representation of the Latin, Cyrillic, Arabic., etc scripts that are used for writing multiple languages

Unicode also distinguishes characters from glyphs, the different forms for the same character - enabling different fonts to be identified as the

Guess That Encoding [1]

Guess That Encoding [2]

\par {\listtext\pard\plain\hich\af0\dbch\af0\loch\f0 1.\tab}}\pard
\ql \fi-
360\li720\ri0\widctlpar\jclisttab\tx720\aspalpha\aspnum\faauto\ls
4\adjustright\rin0\lin720\itap0 {Degree products. The university
currently offers a relatively fixed set of degree \'93 products.\'94
If the university adopted Dell\rquote s \'93build to order\'94
strategy how might its product offerings change? How might
this new strategy affect the university\rquote s \'93market
share\'94 or \'93profitability\'94 compared with its current
strategy?

\par }\pard \ql
\li0\ri0\widctlpar\aspalpha\aspnum\faauto\adjustright\rin0\lin0\it
ap0 {

\par Numerous documents are currently involved in the end-to-

Guess That Encoding [3]

```
<Party>
<Name>Arnold Schwarzenegger/>
<Address>
   <StreetAddress>Governor's Mansion, 1526 H Street</StreetAddress>
   <City>Sacramento</City>
   <State>California</State>
   <PostalCode>95814</PostalCode>
</Address>
<Phone>
   <AreaCode>916</AreaCode>
   <LocalNumber>323-3047</LocalNumber>
</Phone>
</Party>
```

Decoding Web Pages

HTML and XML are easy to decode once you have them, but sometimes what you see in a web page is not what you can get

Web pages can include JavaScript or Flash or other programming components that operate with the source text of the page and substantially modify it before displaying text on the screen

Web pages will also contain text for navigational elements and controls, footers / headers, and various other text you don't want to extract

"Beautiful Soup" is a very useful Python library

Filtering

Removing surrounding header or format information from the text to be processed

What you filter depends on the encoding format or document type

- You'd probably discard HTML markup before indexing
- You'd almost certainly save XML tags for indexing
- You'd probably want to use the rich metadata in email mail headers

Sentence Segmentation

Many IR and text processing applications require that the documents be broken into their constituent sentences

Punctuation marks like -- . , ! ? " -- can make this easy; but not always: sometimes you'll say "Dr. Glushko, this is too hard."

But abbreviations (Dr.) break the obvious rule, and even more complex rules like "period-space-capital letter" signals a sentence break still makes a lot of mistakes

Tokenization into "Wordlike" Elements

Another problem that seems trivial -- just use white space, right?

But what about:

- abbreviations (Dr. is a word)
- hyphens (sometimes part of a word, but sometimes a result of formatting)
- case (do we distinguish Bank from bank)

Preserving Phrases

A phrase is a group of words that needs to be treated as a unit because the meaning of its constituent words can't be combined into the meaning of the phrase

- "birth control" is not "birth" + "control"
- "venetian blinds" are not disabled Italians
- "united states" ...

Phrases can be identified "by hand," using grammatical analysis, and by a "try everything" approach that indexes the candidate phrases that occur with non-negligible frequency

Tokenization Challenges [1]

Character sequences where the tokens include complex alphanumeric structure or punctuation syntax:

- glushko@ischool.berkeley.edu
- 10/26/53
- October 26, 1953
- 55 B.C
- B-52
- 501(c)(3)
- 128.32.226.140
- My PGP key is 324a3df234ch23e
- My home page is http://people.ischool.berkeley.edu/~glushko/

Tokenization Challenges [2]

We've seen that treating numbers and number-like-things as indexable tokens is important

But is 1998 the same as 1,998? Should we ignore the comma?

Is .103 the same as 103? Should we ignore the period?

How many tokens in numeric expressions like 6.022213129 x 10^23 and 6.62606957 ×10−34

How many different numbers might appear in texts? More than the number of word tokens?

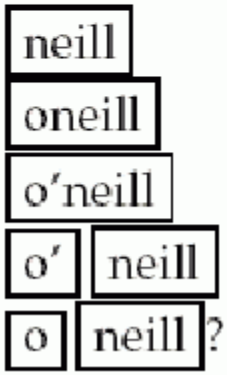
Maybe we should only index numbers when they are combined with letters or punctuation

How could you find a Form 1040?

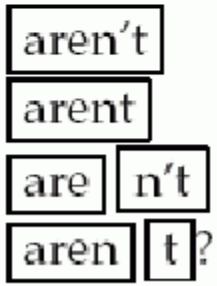
Tokenization Challenges [3]

Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.

For *O'Neill*, which of the following is the desired tokenization?



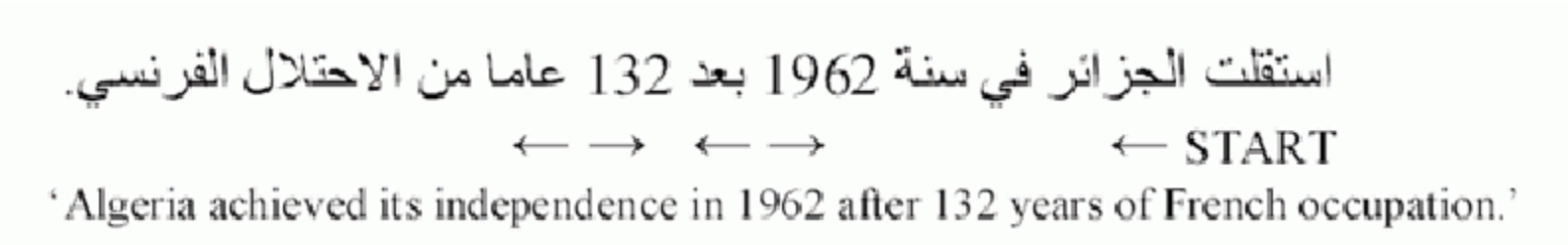
And for *aren't*, is it:



Tokenization Challenges [4]

The language that the characters represent needs to be identified during decoding because it influences the order and nature of tokenization

In languages that are written right-to-left like Arabic and Hebrew, left-to-right text can be interspersed, like numbers and dollar amounts



In German compound nouns don't have spaces between the tokens

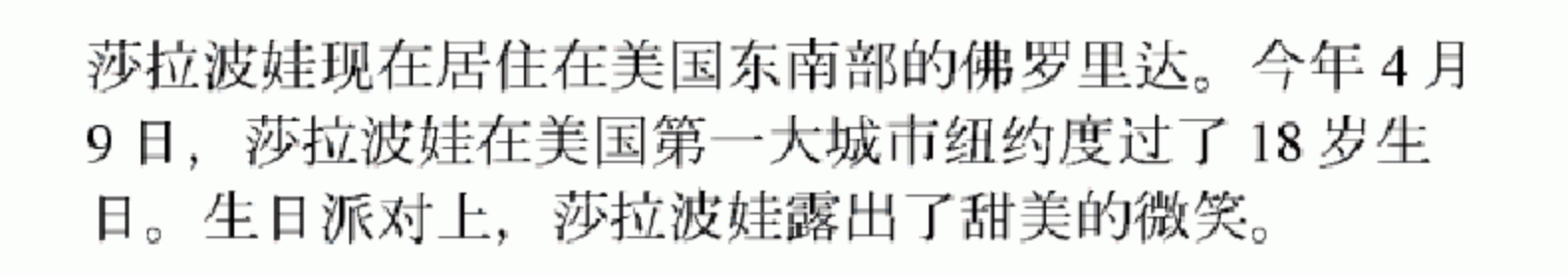
- Lebensversicherungsgesellschaftsangestellter = "life insurance company employee"

Tokenization in "Non-Segmented" Languages

And these problems in "segmented languages" that use white space and punctuation to delimit words seem trivial compared to problems tokenizing the CJK languages that are "non-segmented"

These languages (Chinese, Japanese, and Korean) have ideographic characters that can appear as one-character words but they also can combine to create new words.

The analogous problem in English would be the word "TOGETHER" -- do we treat it as one word or is three separate words "TO GET HER"


莎拉波娃现在居住在美国东南部的佛罗里达。今年 4 月 9 日，莎拉波娃在美国第一大城市纽约度过了 18 岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

Normalization

There will be many cases where two character sequences are not exactly the same but you want to treat them as equivalent

This can be done with "expansion lists" that add query terms at "run time" or by merging the equivalent tokens at "index time"

Capital and lower-case characters are typically "case-folded" to lower-case

Apostrophes, hyphens, accents, and diacritics are often removed (anti-discriminatory and antidiscrimatory, cliché and cliche)

But in most languages diacritics distinguish words (pena and peña mean different things in Spanish)

and @ are taking on additional semantics in hashtags and addresses and in some contexts we'd want to preserve them

One Minute Morphology

MORPHOLOGY is the part of linguistics concerned with the mechanisms by which natural languages create words and word forms from smaller units

These basic building blocks are called MORPHEMES and can express semantic concepts (when they are called ROOTS or abstract features like "pastness" or "plural")

Every natural language contains about 10,000 morphemes and because of how they combine to create words, the number of words is an order of magnitude greater

English has a relatively simple morphological system (http://cla.calpoly.edu/~jrubba/morph/morph.over.html)

Inflection and Derivation

INFLECTION is the morphological mechanism that changes the form of a word to handle tense, aspect, agreement, etc. It never changes the part-of-speech (grammatical category)

- dog, dogs
- tengo, tienes, tenemos, tienen

DERIVATION is the mechanism for creating new words, usually of a different part-of-speech category, by adding a BOUND MORPH to a BASE MORPH

- build + ing -> building; health + y -> healthy

Morphological Processing

Morphological analysis of a language is often used in information retrieval and other low-level text processing applications (hyphenation, spelling correction) because solving problems using root forms and rules is more scalable and robust than solving them using word lists

Natural languages are generative, with new words continually being invented

Many misspellings of common words are obscure low frequency words, so adding them to a misspelling list would make it impossible to check spellings for the latter

Is "Flickr" a misspelled word?

Stemming (1)

STEMMING is morphological processing to remove prefixes and suffixes to leave the root form of words

Stemming reduces many related words and word forms to a common canonical form

This "canonical form" is usually not a word because in the search context this makes it possible to retrieve documents when they contain the meaning we're looking for even if the form of the search word doesn't exactly match what's in the documents

The WordNet stemmer is more conservative, and won't stem a word if doing so creates a text string that isn't in the WordNet dictionary

Stemming (2)

In English, inflectional morphology is relatively easy to handle and "dumb" stemmers (e.g., iteratively remove suffixes, matching longest sequence in rewrite rule) perform acceptably

Stemmers differ in how many affixes they handle; one called Lovins has 260, a more popular stemming algorithm is Porter's Algorithm that is much easier to implement because is has only 60 so it makes a lot more mistakes

Derivational morphology is more difficult

Stemming Mistakes

Stemming affects the recall/precision tradeoff

OVERSTEMMING results when stemming is so aggressive that it reduces words that are not morphologically related to the same root

- Organization, organ
- Policy, police
- Arm, army

UNDERSTEMMING results when stemming is too timid and some morphologically related words are not reduced to the same root

- acquire, acquiring, acquired -> acquir
- acquisition -> acquis

Stop or Noise Words

Any word that doesn't convey meaning by itself can't help us "find out about" anything so it can be discarded during text processing

In English these STOP or NOISE words include:

- determiners, such as "the" and "a(n)"
- auxiliaries, such as "might," "have," and "be"
- conjunctions, such as "and," "that," and "whether"
- degree adverbs, such as "very" and "too"

These words are always among the most frequent in a collection, but high frequency alone isn't what makes them bad index terms

So stop or noise words are usually not determined by frequency analysis -- text processors usually employ a list of them as a kind of negative dictionary

But Stop Words Should be Kept for Phrase Indexing

"President of the United States" is a more precise query than "President" AND "United States"

"To be or not to be"

"Let it Be"

"Flights to London" and "Flights from London" aren't the same query

"Laser printer toner cartridge" vs "Laser printer, with toner cartridge"

Selecting Index Terms

At this stage in text processing the text collection is represented as a set of stems

But not all of them will help a searcher find what they're looking for because they will retrieve too many or too few documents

We can select better index terms if we analyze the distribution of words / stems in the collection

The Index -- Logical View

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Anthony	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	
...							

An index is a data structure that records information about the occurrences of terms in documents

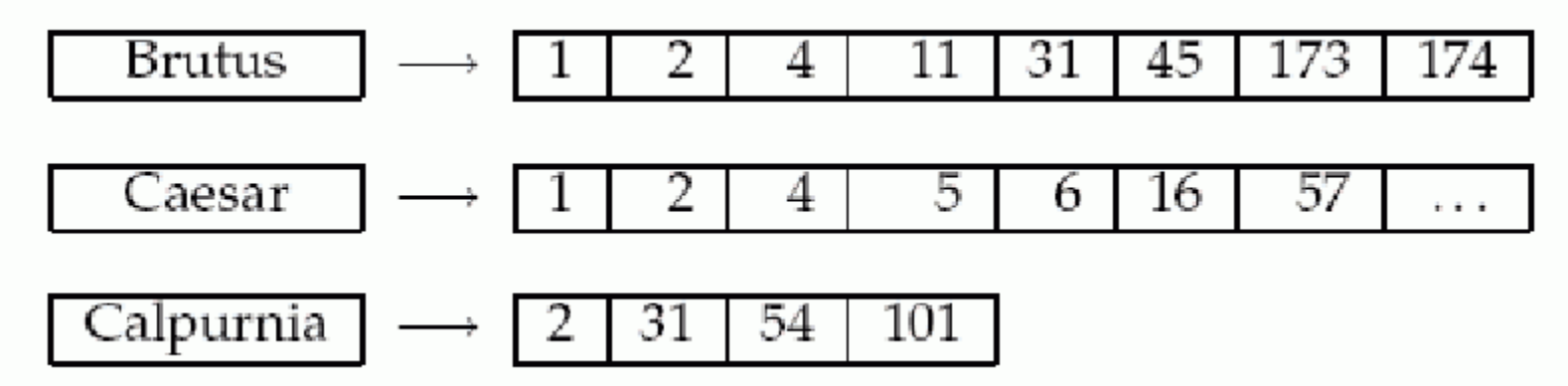
This is a term-document matrix -- rows for terms, columns for documents -- one such data structure

The "Inverted" Index

Using a term-document matrix index representation is both infeasible and nonsensical for any substantial collection of documents

So instead we divide the index into two parts

- A DICTIONARY is a list of the terms
- A POSTINGS LIST is the list of documents in which each term occurs (usually with frequency and position information within each document)



Indexing Step 1 - Term List

Doc 1

Now is the time for all good men to come to the aid of their country

Doc 2

It was a dark and stormy night in the country manor. The time was past midnight

Term	Doc #
now	1
is	1
the	1
time	1
for	1
all	1
good	1
men	1
to	1
come	1
to	1
the	1
aid	1
of	1
their	1
country	1
it	2
was	2
a	2
dark	2
and	2
stormy	2
night	2
in	2
the	2
country	2
manor	2
the	2
time	2
was	2
past	2
midnight	2

Step 2 -- Alphabetize and Merge

Term	Doc #
a	2
aid	1
all	1
and	2
come	1
country	1
country	2
dark	2
for	1
good	1
in	2
is	1
it	2
manor	2
men	1
midnight	2
night	2
now	1
of	1
past	2
stormy	2
the	1
the	1
the	2
the	2
their	1
time	1
time	2
to	1
to	1
was	2
was	2

Term	Doc #	Freq
a	2	1
aid	1	1
all	1	1
and	2	1
come	1	1
country	1	1
country	2	1
dark	2	1
for	1	1
good	1	1
in	2	1
is	1	1
it	2	1
manor	2	1
men	1	1
midnight	2	1
night	2	1
now	1	1
of	1	1
past	2	1
stormy	2	1
the	1	2
the	2	2
their	1	1
time	1	1
time	2	1
to	1	2
was	2	2

Step 3 -- Separate Dictionary and Postings

Term	Doc #	Freq
a	2	1
aid	1	1
all	1	1
and	2	1
come	1	1
country	1	1
country	2	1
dark	2	1
for	1	1
good	1	1
in	2	1
is	1	1
it	2	1
manor	2	1
men	1	1
midnight	2	1
night	2	1
now	1	1
of	1	1
past	2	1
stormy	2	1
the	1	2
the	2	2
their	1	1
time	1	1
time	2	1
to	1	2
was	2	2

Dictionary

Term	N docs	Tot Freq
a	1	1
aid	1	1
all	1	1
and	1	1
come	1	1
country	2	2
dark	1	1
for	1	1
good	1	1
in	1	1
is	1	1
it	1	1
manor	1	1
men	1	1
midnight	1	1
night	1	1
now	1	1
of	1	1
past	1	1
stormy	1	1
the	2	4
their	1	1
time	2	2
to	1	2
was	1	2

Postings

Doc #	Freq
2	1
1	1
1	1
2	1
1	1
1	1
2	1
2	1
1	1
1	1
2	1
1	1
2	1
2	1
1	1
2	1
2	1
1	1
1	1
2	1
2	1
1	2
2	2
1	1
1	1
2	1
1	2
2	2

Boolean Queries

The simplest query language to implement is a Boolean one because it has a very direct correspondence to the text processing story and indexing story we just told

Boolean queries dominate commercial IR systems (and are implemented but rarely used for web searches)

Boolean queries are expressed as Terms + Operators

- Terms are words or stemmed words
- Operators are AND, OR, NOT

Boolean Expressions

Usually expressed with INFIX operators:

- ((a AND b) OR (c AND b))

NOT is UNARY PREFIX operator:

- ((a AND b) OR (c AND (NOT b)))

AND and OR can be n-ary operators:

- (a AND b AND c AND d)

DeMorgan's Law:

- NOT(a) AND NOT(b) = NOT(a OR b)
- NOT(a) OR NOT(b)= NOT(a AND b)

Sample Boolean Queries

Cat

Cat OR Dog

Cat AND Dog

(Cat AND Dog) OR Collar

(Cat AND Dog) OR (Collar AND Leash)

(Cat OR Dog) AND (Collar OR Leash)

Interpreting Boolean Queries

(Cat OR Dog) AND (Collar OR Leash)

Doc #	1	2	3	4	5	6	7
CAT	X	X	X		X	X	
DOG		X	X	X		X	X
COLLAR	X		X	X		X	X
LEASH		X	X		X		X

Doc #	1	2	3	4	5	6	7
CAT		X		X			
DOG	X			X			
COLLAR			X		X		
LEASH			X			X	

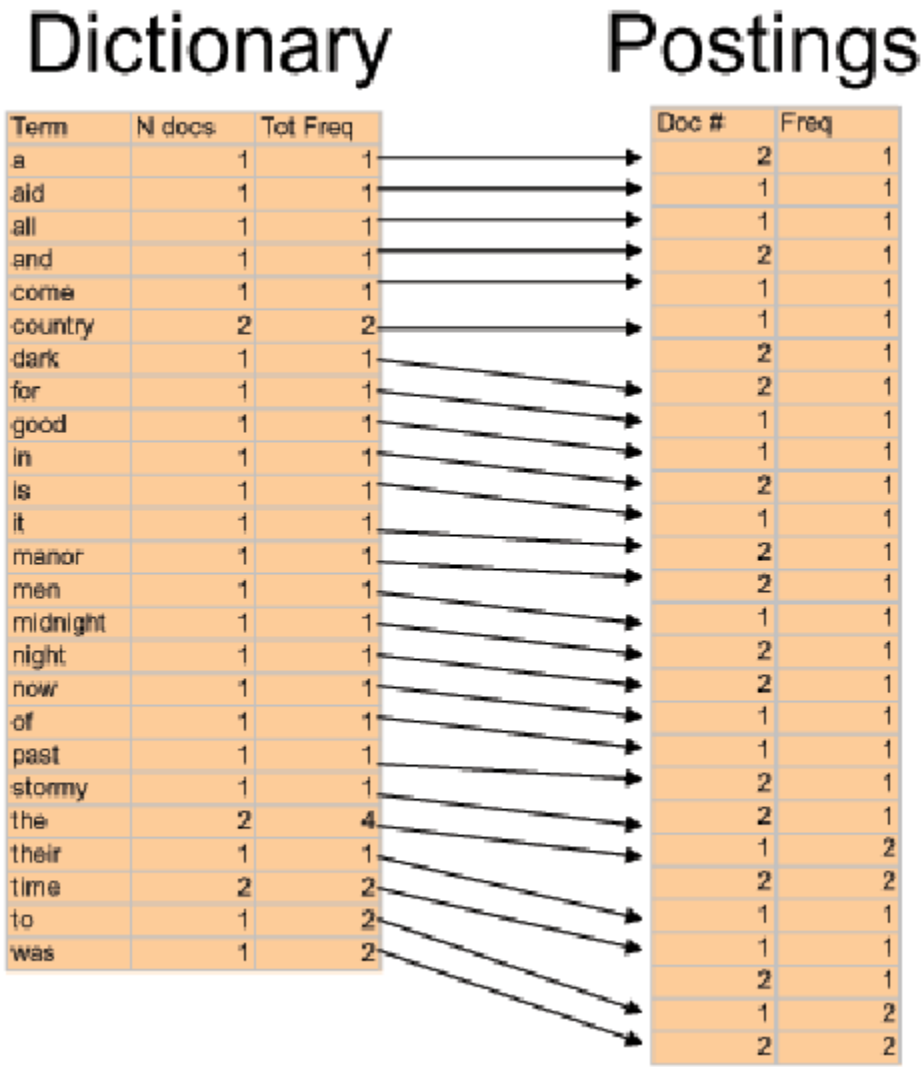
Boolean Search with Inverted Indexes [1]

Permit fast search for individual terms

For each term, you get a list consisting of:

- Document ID
- Frequency of term in doc (optional)
- Position of term in doc (optional)

Boolean Search with Inverted Indexes [2]



Term	N docs	Tot Freq
a	1	1
aid	1	1
all	1	1
and	1	1
come	1	1
country	2	2
dark	1	1
for	1	1
good	1	1
in	1	1
is	1	1
it	1	1
manor	1	1
men	1	1
midnight	1	1
night	1	1
now	1	1
of	1	1
past	1	1
stormy	1	1
the	2	4
their	1	1
time	2	2
to	1	2
was	1	2

Doc #	Freq
2	1
1	1
1	1
2	1
1	1
1	1
2	1
2	1
1	1
1	1
2	1
1	1
2	1
2	1
1	1
2	1
2	1
1	1
1	1
2	1
2	1
1	2
2	2
1	1
1	1
2	1
1	2
2	2

QUERY:
“time AND dark”

2 documents with “time” in dictionary have DOC#1, DOC#2 in posting file

1 document with “dark” in dictionary has DOC#2 in posting file

SOLUTION:
DOC#2 satisfies query

NEXT CLASS MEETING

MIDTERM EXAM

Choice of short-answer questions

Open book, open note, open mind...