

Text Operations

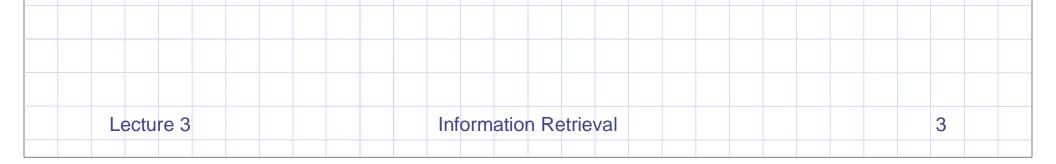
Converting text to indexing terms

- Goal: produce a set of indexing terms
 - that make the best use of resources
- that will accurately match user query terms

| Lecture 3 | Infor | mation Retrieval | 2 |
|-----------|-------|------------------|---|
| | | | |

Text Processing Steps

- 1. Lexical Analysis
- 2. Elimination of stopwords
- 3. Stemming
- 4. Selection of index terms
- 5. Building a thesaurus



Lexical Analysis

Converting byte stream to tokens
a.k.a tokenization or lexing
Three ways to build your lexer
manually (in C or a scripting language)
use a generator such as lex or flex
use a special-purpose DFA generator

Handling of numbers and punctuation should be tunable for the application

Lexing: Numbers and digits

- Numbers need context
 - · "deaths from car accidents in 1989"
 - {deaths, car, accidents, 1989}
 - {1989} could retrieve many irrelevant docs
- · However...
 - numbers do appear in user queries
 - rest of terms can give context
 - might be helped by using phrases

Lexing: Hyphens

- Keep them?
 - · query might use a non-hyphenated variant
 - end-of-line hyphens are noise
- Throw them out?
 - · can't recognize a hyphenated term in a query
- Two advanced solutions
 - index as phrase but allow partial matches
 - use proximity information

Lexing: Punctuation

Obvious: segment on puctuation But (like hyphens) can appear inside a single term:

 "B.C.", "B.S.": without periods, these are just single letters

- URLs as index terms?
- Idea: look at surrounding characters
 - whitespace? end of sentence
 - not whitespace? abbreviation

Lexing: Markup

- Nowadays, everything has markup
 - · SGML, HTML, XML...
 - · This information can be useful or not...
- Some alternatives:
 - · emit text appearing inside all or some tags
 - emit tags as tokens which can be interpreted by the indexer.

Writing a lexer by hand

- while ((c = getchar()) != EOF)
 - if (isalpha(c)) { ...
- Very fast! but
 - Error-prone
 - · Hard to make it flexible or modular
 - Alternative: use a scripting langauge
 - Easier to describe text patterns
 - But can be hard to maintain

Using a DFA generator

Generalization of the hand-written lexer Define a state machine transitions occur on different character input • states define possible next steps \cdot write a table, not a procedure Program generates the lexer Easier to maintain and debug! (Frakes & Baeza-Yates '92 have code)

Stop Words

- the, of, and, a, in, to, is, for, with, are
 - take up a lot of space
 - · retrieve all documents
 - don't relate to information need
- It's easy to index something that appears everywhere
 - Removing stopwords can cause problems:
 - "to be or not to be" \rightarrow {be}
 - "C" as a stop word would be trouble for a computer programming index!

Removing Stop Words

Start with a list of stop words Table lookup Make a table out of a static stoplist Match each token against the table Hashes, perfect hashing, tries • Build into the lexical analyzer (see F&BY) Or take a statistical approach

Stemming

Reduce variant word forms to a single "stem" form

- -'s, -ing, -ed, -s; in-, ad-, pre-, sub-, ...
 - Four approaches
- table lookup use a dictionary
 - successor variety fancy suffix removal
- affix removal cut prefixes and suffixes
- character n-grams (not really stemming)

Porter's algorithm (1980)

Stage 1a and b

SSES -> SS caresses -> caress Removes suffixes in • IES -> I | ponies -> poni five stages ties -> ti Only one rule in each SS -> SS | caress -> caress stage fires Each depends on a $S \rightarrow \phi \mid cats \rightarrow cat$ • suffix and the stem (m>0) feed -> feed EED->EE agreed -> agree measure *m* $[C](VC)^{m}[V]$ (*v*) ED-> | plastered -> plaster (*v*) ING-> motoring -> motor Lecture 3 Information Retrieval 14

Porter Errors (Krovetz 93)

- Too eager
- organization/organ
- doing/doe
 - policy/police
- university/universe
- negligible/negligent
 - arm/army
- past/paste

- Too cautious
- european/europe
- matrices/matrix
 - create/creation
 - machine/machinery
- · explain/explanation
- resolve/resolution
- triangle/triangular

Stems and roots

Stemmers are language specific

See the Snowball project
http://snowball.sourceforge.net/
for stemmers in other languages

- Morphological analysis
 - reducing words to their linguistic roots
 - requires more sophisticated processing
 - Think about how this can affect the query

Character n-grams

- Slide an *n*-character window through text No stemming or stoplisting May need to consider punctuation and hyphens
- Redundant tokens: good for noisy text
- Less effective than word (stem) pairs in clean text

Term Selection

Individual words Adjacent word pairs (word n-grams) Noun phrases requires more sophisticated NLP identify nouns along with adjectives and adverbs in the same phrase "computer science" and "world-wide web" •

The Case for Complexity

User queries are only one or two words The bag-of-words approach is too simplistic given short queries

 Using phrases, sophisticated handling for numbers, etc. boosts the quality of that first list of documents.

The Case for Simplicity

- Query throughput is as (more?) important than quality responses
- Disk is cheap
- Complex processing takes too long
- Easy to make a wrong decision
 - Feedback will improve the results

| Lecture 3 |
|-----------|
|-----------|

Simple or Complex?

Can look at it on two levels: Does more sophisticated term processing improve retrieval results? ... or ... Does it enable a more sophisticated interface for the user?

Lecture 3

Information Retrieval

Designing with Filters

- The UNIX philosophy: "do one thing and do it well."
- Filters read text input and produce text output
 - · can be linked together in pipes
 - · can be simple (cut, nl) or complex (awk,perl)
 - Lexers are filters
 - You can have several in your toolbox