

Token Identification

1

Token Processing

- Parser Generators
- Stemming
- n-grams
- Phrase Recognition

2

Stemming

- Goal of stemming is to improve effectiveness by working to ensure that mismatches in prefixes and suffixes do not cause a miss of a query and a relevant document.
- User who is searching for “swimming” might be interested in documents with “swim”
 - Algorithms
 - Porter
 - Dictionary-based
 - K-stem
 - Others

3

Porter Stemmer

- An incoming word is cleaned up in the initialization phase, one prefix trimming phase then takes place and then five suffix trimming phases occur.
- Note: The entire algorithm will not be covered -- we will leave out some obscure rules.

4

Initialization

- First the word is cleaned up. Converted to lower case only letters or digits are kept.
- F-16 is converted to F16.

5

Porter Stemming

- Remove prefixes:
"kilo", "micro", "milli", "intra", "ultra",
"mega", "nano", "pico", "pseudo"

So megabyte, kilobyte all become “byte”.

6

Porter Step 1

- Remove “es” from words that end in “sses” or “ies”
 - passes -> pass, cries -> cri
- Remove “s” from words whose next to last letter is not an “s”
 - runs -> run, fuss -> fuss
- If word has a vowel and ends with “eed” remove the “ed”
 - agreed -> agree, freed -> freed
- Remove “ed” and “ing” from words that have no other vowel
 - dreaded -> dread, red -> red, bothering -> bother, bring -> bring
- Add “e” if word has a vowel and ends with “ated” or “bled”
 - enabled -> enable, generated -> generate
- Replace trailing “y” with an “i” if word has a vowel
 - satisfy -> satisfi, fly -> fly

7

Porter Step 2

- With what is left, replace any suffix on the left with suffix on the right

...

tional	tion	conditional -> condition
ization	ize	nationalization -> nationalize
iveness	ive	effectiveness -> effective
fulness	ful	usefulness -> useful
ousness	ous	nervousness -> nervous
ousli	ous	nervously -> nervous
entli	ent	fervently -> fervent
iveness	ive	inventiveness -> inventive
biliti	ble	sensibility -> sensible

8

Step 3

- With what is left, replace any suffix on the left with suffix on the right

...

icate	ic	fabricate -> fabric (<i>Think about this one</i>)
ative	--	combativ -> comb (<i>another good one</i>)
alize	al	nationalize -> national
iciti	ic	
ical	ic	tropical -> tropic
ful	--	faithful -> faith
iveness	ive	inventiveness -> inventive
ness	--	harness -> har

9

Step 4

- Remove remaining standard suffixes
al, ance, ence, er, ic, able, ible, ant, ement, ment, ent, sion, tion, ou, ism, ate, iti, ous, ive, ize, ise

10

Step 5

- Remove trailing “e” if word does not end in a vowel
 - hinge -> hing
 - free -> free

11

Porter Summary

- Pro
 - tends to improve effectiveness a few percent
- Con
 - many words with different meanings have common stems (e.g.; *fabricate* and *fabric*)
 - a lot of stems are not words

12

Dictionary based approaches

- Examine word for common endings
- Develop some candidate words by removing the endings
- Find the longest word that is in the dictionary that matches one of the candidates.
- Pro
 - This eliminates the Porter problem that many stems are not words.
- Con
 - Language dependent approach

13

K-stem

- Start with Porter or Dictionary based stemmer
- Place words in potential classes
- Measure the frequency of co-occurrence of terms in the class
- Eliminate words from a class with a low co-occurrence
- Remaining classes form stemming rules

14

K-stem

- Pro
 - Language independent
 - Based on assumption that terms in a class will co-occur with other terms “hippo” will co-occur with “hippos”
 - Improves effectiveness
- Con
 - computationally expensive to build co-occurrence matrix (but you only do it every now and then)

15

Parser Generators

- Goal is to allow users to specify parsing rules as grammars.
- Grammars provide a very flexible means of expressing all valid strings in a language.
 - Examples of tools that do this
 - Lexx / Yacc
 - Javacc

16

Grammar

- A language can be defined as a set of valid strings.
- A grammar is merely a set of rules that provide a means of expressing all the valid strings in a language.

17

Example

- Consider the language of all strings that start with some number of a 's and are then followed by the same number of b 's.

$L = a^n b^n$

valid strings are “aaabbb”, “ab”, “aabb”

$S \rightarrow a A b \mid \lambda$

$A \rightarrow a S b \mid ab$

λ is the empty string.

18

Grammar

- A grammar can be written easily for program languages so parsers for a compiler can be generated.
- Regular expressions can be used to indicate string patterns.
- a^+ -- one or more a's
- a^* -- zero or more a'

19

Some useful regular expressions

Acronym: $(["A"- "Z"] (["A"- "Z"])^*)^*$ Ex: NCR, IBM, etc.

Abbreviation: $(["A"- "Z"] ".")^*>$ Ex: U.S.A.

Model: $["a"- "z", "A"- "Z"] " ("["0"- "9"])^*>$ Ex: F-16, C-25

Word: $["a"- "z", "A"- "Z"] (["a"- "z", "A"- "Z"])^*>$ Ex: hippo, Hippo

Integer: $["0"- "9"] (["0"- "9"])^*>$ Ex: 123

Decimal: $(["0"- "9"])^* "." (["0"- "9"])+>$ Ex: 123.45

20

Parsing a File

File consists of zero or more documents followed by an EOF marker.

$S \rightarrow \langle \text{document} \rangle \mid \text{EOF}$.

void Input() :

```
int DocId = 0;
Document d = new Document(0);

(parseDocument(d)

    d.output();
    ++DocId;
    d = new Document(DocId);

)* -EOF>
```

21

Parsing a Document

- Document consists of an optional headline or dateline followed by some text.
- $\langle \text{document} \rangle \rightarrow \langle \text{headline} \rangle \mid \langle \text{dateline} \rangle \mid \langle \text{textbegin} \rangle$
- $\langle \text{textbegin} \rangle \rightarrow \langle \text{text_begin} \rangle \langle \text{word} \rangle \langle \text{text_end} \rangle$
- $\langle \text{word} \rangle \rightarrow \langle \text{model} \rangle \mid \langle \text{acronym} \rangle \mid \langle \text{integer} \rangle$, etc.

22

N-grams

- Terms are all strings of length n
- Language-independent -- no stemming or stop word removal needed
- Tolerates noise, misspelled words

23

5-Gram Example

- Q: What technique **works on noise** and **misspelled** words?
- D_1 : N-grams **work on noisy misspelled** text.

_work	spell	• 8 terms are matched
_on_no	pelle	• No stemming of work, noise
on_noi	elled	• Partial match of misspelled
n_nois	lled_	word

24

N-gram Summary

- Pro
 - Language independent
 - Works on garbled text (OCR, etc.)
- Con
 - there can be a LOT of n-grams, dictionary may not fit in memory anymore
 - query processing requires more resources

25

Phrase processing

- Phrase recognition is based on the goal of indexing meaningful phrases like
 - “Lincoln Town Car”
 - “San Francisco”
 - “apple pie”
- Doing this would use word order to assist with effectiveness -- otherwise we are assuming the query and documents are just a “bag of words”

26

Phrase Processing

- Statistical
 - start with all 2-word pairs that are not separated by punctuation or stop words
 - only keep those that occur more than x times
- Natural Language
 - do a full parse of the sentence (S-V-OBJ)
 - keep all noun phrases “Republic of China”

27

Modifying the Query

- Phrase Processing Options
 - Add phrase terms to the query just like other terms
 - This really violates independence assumptions but a lot of people do it anyway
 - Give phrase terms a different weight than query terms

28

Phrase Processing Summary

- Pro
 - Often found to improve effectiveness by 10 percent
- Con
 - Dramatically increases size of term dictionary and the size of the index

29

Parsing Summary

- Parsing can make a difference in effectiveness
- Parsing is often overlooked
- Language independence is preferred

30