

Multimedia Information Retrieval and Technology

L3. Vocabulary

By : Fangyu Wu

Room: SD555



Xi'an Jiaotong-Liverpool University

西交利物浦大學

Precision and recall

	Relevant	Nonrelevant
Retrieved	true positives (TP)	false positives (FP)
Not retrieved	false negatives (FN)	true negatives (TN)

$$P = TP / (TP + FP)$$

$$R = TP / (TP + FN)$$

A collection with 1000 docs , 100 are relevant to the query. System AAA retrieved 80 docs, while 20 docs are relevant. What are P and R, respectively?

Solution:

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

	relevant 100	non-relevant.
retrieved 80	20 TP	60 FP
not retrieved	80 FN	840 TN

$$\therefore P = \frac{20}{20 + 60} = \frac{20}{80} = \frac{1}{4}$$

$$R = \frac{20}{20 + 80} = \frac{20}{100} = \frac{1}{5}$$

Precision and recall

- Precision (P) is the fraction of retrieved documents that are relevant

$$\text{Precision} = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})} = P(\text{relevant}|\text{retrieved})$$

- Recall (R) is the fraction of relevant documents that are retrieved

$$\text{Recall} = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})} = P(\text{retrieved}|\text{relevant})$$

- Positional postings and phrase queries
- Document delineation and character sequence decoding
- The vocabulary of terms

Phrase queries

We want to be able to answer queries such as
“***stanford university***” – as a phrase

Thus the sentence “*I went to university at Stanford*”
is not a match.

- Many *implicit phrase queries*: San Francisco or people's name

It no longer suffices to store only
<term: doc ID> entries



A first attempt: Biword indexes

- Index every consecutive pair of terms in the text as a phrase
- For example the text “Friends, Romans, Countrymen” would generate the biwords
 - ***friends romans***
 - ***romans countrymen***
- Each of these biwords is now a dictionary term;
- Two-word phrase/query processing is now feasible.

Longer phrase queries

Longer phrases can be processed by breaking them down

Queries like “***stanford university palo alto***”
can be broken into the Boolean query on biwords:
***stanford university AND university palo AND palo
alto***

-- the docs matching the above Boolean query may
not contain the original phrase.



Can have false positives!

Issues for biword indexes

- False positives, as noted before
- Index blow-up due to bigger dictionary
 - Infeasible for more than biwords, bigger even for them

Phrase queries:

Solution1: Biword indexes

Solution2: Positional indexes

Solution 2: Positional indexes

For each ***term*** in the postings, store the position(s) in which tokens appear:

<***term***, number of docs containing term;

doc1: position1, position2 ... ;

doc2: position1, position2 ... ;

etc.>



Solution 2: Positional indexes

Rather than simply checking that both terms are in a document, you also need to check that their positions of appearance are compatible with the phrase query being evaluated.

Working out offsets between the words.

Positional index example

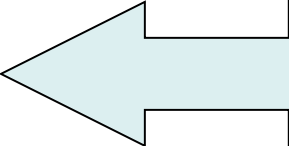
<*be*: 993427;

1: 7, 18, 33, 72, 86, 231;

2: 3, 149;

4: 17, 191, 291, 430, 434;

5: 363, 367, ...>



Which of docs *1,2,4,5*
could contain “*to be*
or not to be”?

Processing a phrase query

Step1: Extract inverted index entries for each distinct term: ***to, be, or, not.***

Step2: Merge their *doc:position* lists to enumerate all positions with “***to be or not to be***”.

to:

2:1,17,74,222,551;

4:8,16,190,429,433;

7:13,23,191; ...

be:

1:17,19;

4:17,191,291,430,434;

5:14,19,101; ...

Positional index size

A positional index expands postings storage *substantially*.

- Even though indices can be compressed.

Nevertheless, a positional index is now more standardly used.

Exercise

Shown below is a portion of a positional index in the format: term: doc1: ⟨position1, position2, ...⟩; doc2: ⟨position1, position2, ...⟩; etc.

angels: 2: ⟨36,174,252,651⟩; 4: ⟨12,22,102,432⟩; 7: ⟨17⟩;
fools: 2: ⟨1,17,74,222⟩; 4: ⟨8,78,108,458⟩; 7: ⟨3,13,23,193⟩;
fear: 2: ⟨87,704,722,901⟩; 4: ⟨13,43,113,433⟩; 7: ⟨18,328,528⟩;
in: 2: ⟨3,37,76,444,851⟩; 4: ⟨10,20,110,470,500⟩; 7: ⟨5,15,25,195⟩;
rush: 2: ⟨2,66,194,321,702⟩; 4: ⟨9,69,149,429,569⟩; 7: ⟨4,14,404⟩;
to: 2: ⟨47,86,234,999⟩; 4: ⟨14,24,774,944⟩; 7: ⟨199,319,599,709⟩;
tread: 2: ⟨57,94,333⟩; 4: ⟨15,35,155⟩; 7: ⟨20,320⟩;
where: 2: ⟨67,124,393,1001⟩; 4: ⟨11,41,101,421,431⟩; 7: ⟨16,36,736⟩;

Which document(s) if any meet each of the following queries, where each expression within quotes is a phrase query?

a. “fools rush in”

2, 4, 7

b. “fools rush in” AND “angels fear to tread”

4



Xi'an Jiaotong-Liverpool University

西交利物浦大学

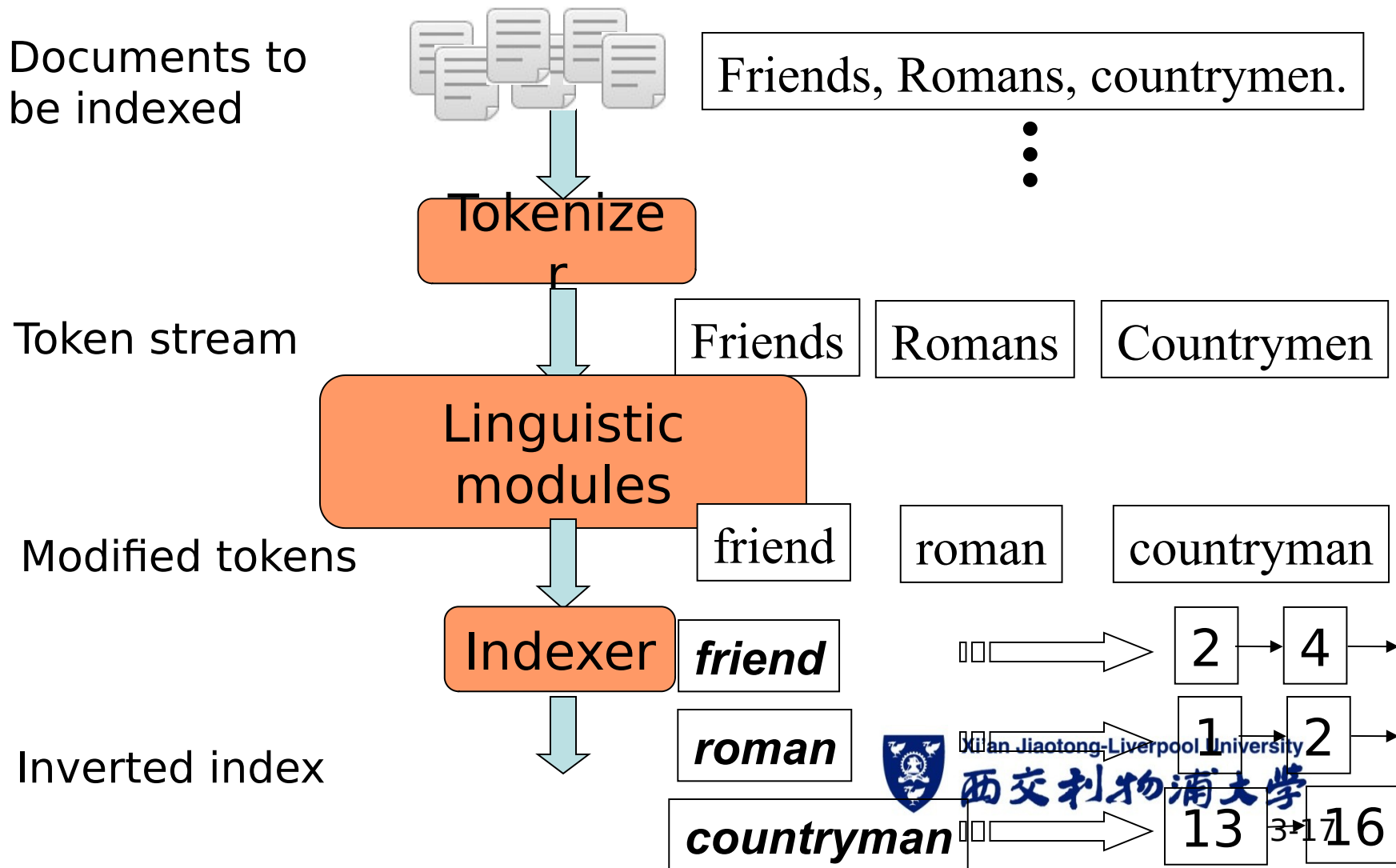
Solution

Answer (a): All three documents (2, 4, and 7) satisfy the query.

Answer (b): Only document 4.

- Positional postings and phrase queries
- Document delineation and character sequence decoding
- The vocabulary of terms

Recall the basic indexing pipeline



Terms and documents

- Last lecture: Simple Boolean retrieval system
- Our assumptions were:
 - We know what a document is.
 - We know what a term is.
- Both issues can be complex in reality.
- We'll look a little bit at what a document is.
- But mostly at terms: How do we define and process the vocabulary of terms of a collection?

Parsing a document

Digital documents that are the input to an indexing process are typically **bytes** in a file or on a web server.

- What format is it in?
pdf/word/excel/html?
- What language is it in?
- What character set is in use?
(CP1252, UTF-8, ...)

Determine the correct encoding. Once the encoding is determined, we decode the byte sequence to a character sequence.

Complications: Format/language

Documents being indexed can include many different languages

- A single index may contain terms from many languages.
- French email with a German pdf attachment.
- French email quote clauses from an English-language contract
- Chinese paper with English abstract or quote.

Complications: What is a document?

What is a unit document?

- A file?
- An email?
- What about an email with 5 attachments?
- A group of files (e.g., PPT or LaTeX split over HTML pages)

Document Unit

Normally, we take each file in a folder as a document;
An email file stores a sequence of email messages in one file. But you might wish to regard each email message even each attachment as separate documents.

Opposite direction, various web software will split PPT or LATEX document into separate HTML pages for each slide or subsection, stored as separate files. Where we might want to regard them as a single document.

Document Unit

For a collection of books, it would usually be a bad idea to index an entire book as a document.

- index each chapter or paragraph as a mini-document.

Treat individual sentences as mini-documents?

It becomes clear that there is a precision/recall tradeoff here.

Document Unit

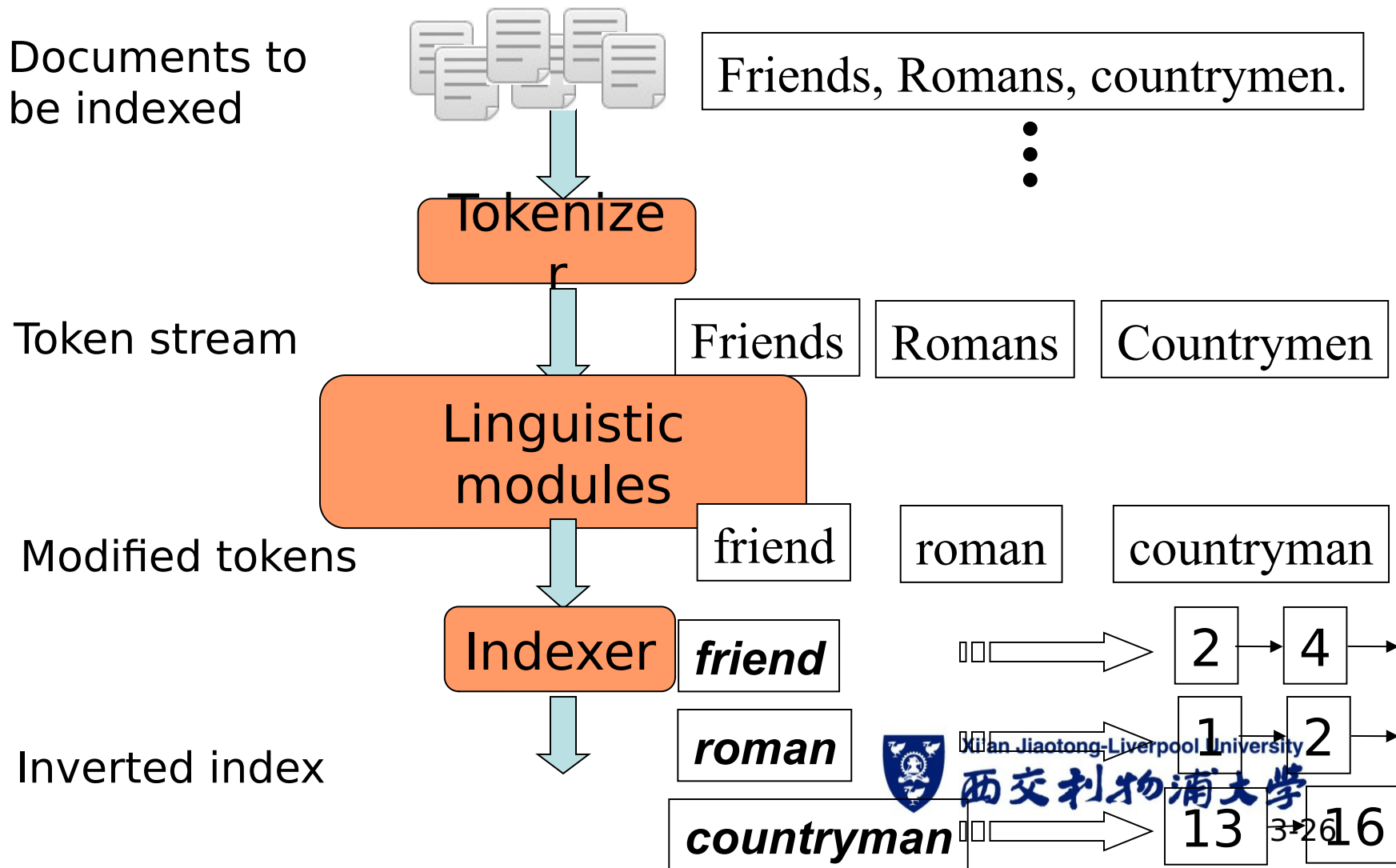
If the units get **too small**, we are likely to miss important passages because terms were distributed over several mini-documents;

While if units are **too large**, we tend to get spurious matches and the relevant information is hard for the user to find.

Determining the vocabulary of terms

- Tokenization
- Stop words
- Normalization
- Stemming and Lemmatization

Recall the basic indexing pipeline



Basic Concept

Tokenization: the process of chopping character streams into tokens;

Linguistic preprocessing then deals with building equivalence classes of tokens, which are the set of terms that are indexed.

Tokenization

Input: “***Friends, Romans and Countrymen***”

Output: Tokens

Friends

Romans

Countrymen

Each such token is now a candidate for an index entry, after further processing

But what are valid tokens to emit?

Tokenization

What are the correct tokens to use?

Normally, you chop on whitespace and throw away punctuation characters.

A number of tricky cases.

Tokenization

E.g. : what do you do about the various uses of the apostrophe for possession and contractions:

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

neill	
oneill	
o’neill	
o’	neill
o	neill?

aren’t	
arent	
are	n’t
aren	t?

Language Identification

These issues of tokenization are **language-specific**.
It thus requires the language of the document to be known.

Language identification based on classifiers that use short character subsequences as features is highly effective;

Most languages have distinctive signature patterns

Specific tokens

Programming languages: C++, C#

Aircraft names: B-52

A television show: M*A*S*H

Email addresses

Web URLs (<http://www.baidu.com/>)

IP addresses (142.32.48.231)

Package tracking numbers

Issues in tokenization

Hewlett-Packard → ***Hewlett*** and ***Packard*** as two tokens?

state-of-the-art: break up hyphenated sequence.

co-education

lowercase, lower-case, lower case ?

Handling hyphens automatically is complex: it can either be done as a classification problem, or more commonly by some heuristic rules, such as allowing short hyphenated prefixes on words, but not longer hyphenated forms.

Issues in tokenization

An effective method: enter hyphens wherever they may be possible

whenever there is a hyphenated form, the system will generalize the query **to cover all of the three forms**, so that a query for over-eager will search for over-eager OR “over eager” OR overeager.

However, this strategy depends on user training, since if query using either of the other two forms, you get no generalization

Issues in tokenization

Splitting on white space can also split what should be regarded as a single token.

New York: one token or two?

Other cases with internal spaces that we might wish to regard as a single token include phone numbers ((800) 234-2333) and dates (Mar 11, 1983).

Tokenization: language issues

Chinese and Japanese have no spaces between words:

Not always guaranteed a unique tokenization

Further complicated in Japanese, with multiple alphabets intermingled

Dates/amounts in multiple formats

フォーチュン 500 社は情報不足のため時間あた \$500K(約 6,000 万円

Katakana Hiragana Kanji Romaji

End-user can express query entirely in hiragana!

Tokenization: language issues

To perform *word segmentation* as prior linguistic processing.

1. Chinese character is more like a syllable than a letter and usually has some semantic content.
2. Most words are short, the commonest length is two characters
3. Lack of standardization of word breaking in the writing system.

Ambiguous segmentation in Chinese



The two characters can be treated as one word meaning 'monk' or as a sequence of two words meaning 'and' and 'still'.

Tokenization: language issues

استقلت الجزائر في سنة 1962 بعد 132 عام من الاحتلال الفرنسي.
 ← → ← → ← start

‘Algeria achieved its independence in 1962 after 132 years of French occupation.’

With Unicode, the surface presentation is complex,
 but the stored form is straightforward
 (Bidirectionality is not a problem if text is coded in
 Unicode.)

Determining the vocabulary of terms

- Tokenization
- Stop words
- Normalization
- Stemming and Lemmatization

Stop words

Stop words: Common words that would be of **little value** in helping select documents matching a user need are excluded from the vocabulary.

Intuition:

- They have little semantic content: *the, a, and, to, be*
- There are a lot of them: ~30% of postings for top 30 words

Stop words

But the trend is away from doing this:

Good compression techniques means the space for including stop words in a system is very small

Good query optimization techniques mean you pay little at query time for including stop words.

You need them for:

- Phrase queries: “King of Denmark”
- Various song titles, etc.: “Let it be”, “To be or not to be”
- “Relational” queries: “flights to London”

Determining the vocabulary of terms

- Tokenization
- Stop words
- **Normalization**
- Stemming and Lemmatization

Normalization to terms

- Having broken up our documents (and also our query) into tokens, the easy case is if tokens in the query just match tokens in the token list of the document.
- We may need to “normalize” words in indexed text as well as query words into the same form
E.g. We want to match **U.S.A.** and **USA**

Normalization to terms

Token normalization: *the process of normalizing tokens so that matches occur despite differences in the character sequences of the tokens.*

Result is terms: **a term** is a (normalized) word type, which is an entry in our IR system dictionary.

Normalization (equivalence classing)

We most commonly define **equivalence classes** of terms by, e.g.,

- deleting periods to form a term

U.S.A., USA (USA)

- deleting hyphens to form a term

***anti-discriminatory, antidiscriminatory
(antidiscriminatory)***

If the tokens U.S.A. and USA are both mapped onto the term USA, in both the document text and queries, search for one term will retrieve documents that contain either.

Normalization

An alternative way: maintain relations between unnormalized tokens, (this method can be extended to **hand-constructed lists of synonyms** such as *car* and *automobile*)

These term relationships can be achieved in two ways.

- Query expansion
- To perform the expansion during index construction

Normalization

Query Expansion

to index unnormalized tokens and to maintain a query expansion list of multiple vocabulary entries to consider for a certain query term.

Normalization

Index Expansion

To perform the expansion during index construction.

When the document contains automobile, we index it under car as well (and, usually, also vice-versa).

Normalization

Less efficient than equivalence classing, as there are more postings to store and merge.

- **Query Expansion**
- **Index Expansion**

The first method adds a query expansion dictionary and requires more processing at query time, while the second method requires more space for storing postings.

Normalization

Asymmetric expansion: Powerful and flexible, but less efficient

Query term	Matched terms in documents
Windows	Windows
windows	Windows,windows>window
window	window,windows

The expansion lists can overlap while not being identical

Normalization: Accents

Accents: e.g., French résumé vs. resume.

- Should be equivalent

Most important criterion:

- How are your users like to write their queries for these words?

Even in languages that standardly have accents, users often may not type them - best to normalize to a de-accented term.

Normalization: Case folding

Reduce all letters to lower case.

Exception: upper case in mid-sentence? (many proper nouns are derived from common nouns only by case)

- Companies: e.g., General Motors
- Government organizations: Fed (the Federal Reserve System) vs. fed
- Person names: e.g., Bush, Black

Often best to lower case everything, since users will use lowercase regardless of ‘correct’ capitalization...

Normalization: other issues

You might wish to equate the British spelling *colour* and the American spelling *color*.

Normalization of things like date forms

7 月 30 日 vs. 7/30 vs. 30th, July

However, correct processing here is complicated by the fact that in the U.S., 3/12/91 is *Mar. 12, 1991*, whereas in Europe it is *3 Dec 1991*.

Normalization: other issues

Tokenization and normalization may depend on the language and so is intertwined with language detection.

Crucial: “normalize” indexed text as well as query terms **identically**.

Determining the vocabulary of terms

- Tokenization
- Stop words
- Normalization
- Stemming and Lemmatization



Stemming and Lemmatization

Goal: to reduce inflectional/variant forms to a base form.

am, are, is → be

car, cars, car's, cars' → car

- Different forms of a word, E.g., organize, organizes, and organizing.
- Derivationally related words, e.g., democracy, democratic, and democratization.

E.g., the boy's cars are different colors → the boy car be different color

Stemming and Lemmatization

However, the two words differ in their flavor.

Stemming usually **chops off** the ends of words in the hope of achieving this goal correctly and often includes the removal of derivational affixes.

Lemmatization refers to doing things properly with the use of a vocabulary and morphological analysis of words, aiming to remove inflectional endings only, and to **return the base or dictionary form of a word**, which is known as the *lemma*.

Stemming and Lemmatization

E.g., when tokenize organizing, organizes,

Stemming: organiz

Lemmatization: organize

If confronted with the token saw,

Stemming: s

Lemmatization: see or saw

Stemming

Reduce terms to their “roots” before indexing

“Stemming” suggests crude affix chopping (chops off the ends of a word)

language dependent

e.g., *automate(s)*, *automatic*, *automation* all reduced to *automat*.

for example compressed and compression are both accepted as equivalent to compress.



for exampl compress and compress ar both accept as equival to compress



Xi'an Jiaotong-Liverpool University

西交利物浦大学

Stemming and Lemmatization

Linguistic processing for stemming or lemmatization is often done by an additional plug-in component to the indexing process, a number of such components exist both commercial and open source.

Porter's algorithm

Commonest algorithm for stemming English

sses → *ss*

ies → *i*

ational → *ate*

tional → *tion*



Typical rules in Porter

- Weight of word sensitive rules:

$(m > 1)$ *EMENT* →

E.g.:

replacement → *replac*

cement → *cement*

(Delete final *EMENT* if what remains is longer than 1 character)

Does stemming help?

English: very mixed results. Helps recall for some queries but harms precision on others

E.g., operative \Rightarrow oper

Definitely useful for Spanish, German, Finnish, ...

30% performance gains for Finnish!

Stemming and Lemmatization

Stemmers use language-specific rules, but they require less knowledge than a lemmatizer, which needs a complete vocabulary and morphological analysis to correctly lemmatize words.

Exercise

	Relevant	Nonrelevant
Retrieved	TP	FP
Not retrieved	FN	TN

$$P = \frac{TP}{TP + FP}$$

$$(b) = \frac{TP}{TP + FN}$$

Are the following statements true or false? When false please give the true statements

- a) In a Boolean retrieval system, stemming never lowers precision. (F.)
may generate some more FPs.
- b) In a Boolean retrieval system, stemming never lowers recall. (T.)
- c) Stemming increases the size of the vocabulary. (F.)
- d) Stemming should be invoked at indexing time but not while processing a query. (F.)



Solution

- a. False. Stemming can increase the retrieved set without increasing the number of relevant documents.
- b. True. Stemming can only increase the retrieved set, which means increased or unchanged recall.
- c. False. Stemming decreases the size of the vocabulary.
- d. False. The same processing should be applied to documents and queries to ensure matching terms.