More on indexing and text operations

CE-324: Modern Information Retrieval

Sharif University of Technology

M. Soleymani

Fall 2018

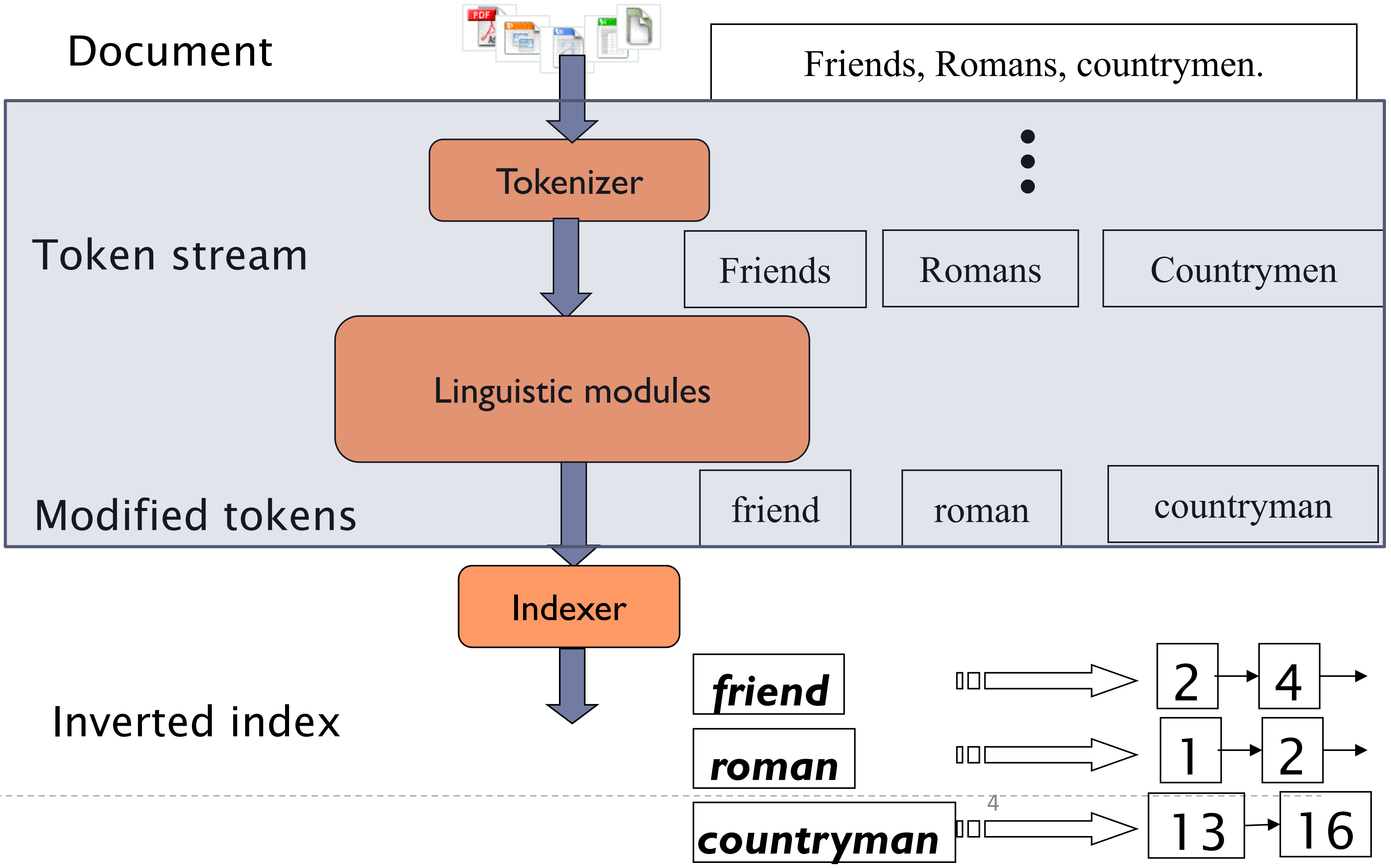
Most slides have been adapted from: Profs. Manning, Nayak & Raghavan (CS-276, Stanford)

Plan for this lecture

- Text operations: Preprocessing to form the term vocabulary
- Elaborate basic indexing
  - Positional postings and phrase queries

Text operations & linguistic preprocessing

Recall the basic indexing pipeline



Text operations

- Tokenization
- Stop word removal
- Normalization
  - Stemming or lemmatization
  - Equivalence classes
    - Example1: case folding
    - Example2: using thesauri (or Soundex) to find equivalence classes of synonyms and homonyms

Parsing a document

- What format is it in?
  - pdf/word/excel/html?
- What language is it in?
- What character set is in use?

These tasks can be seen as classification problems, which we will study later in the course.

But these tasks are often done heuristically …

Indexing granularity

- What is a unit document?
  - A file? Zip files?
  - Whole book or chapters?
  - A Powerpoint file or each of its slides?

Tokenization

- Input: "***Friends, Romans, Countrymen***"
- Output: Tokens
  - ***Friends***
  - ***Romans***
  - ***Countrymen***

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

Tokenization

- Issues in tokenization:
  - ***Finland's capital*** → ***Finland? Finlands? Finland's***?
  - ***Hewlett-Packard*** → ***Hewlett*** and ***Packard*** as two tokens?
    - ***co-education***
    - ***lower-case***
    - ***state-of-the-art***: break up hyphenated sequence.
    - It can be effective to get the user to put in possible hyphens
  - ***San Francisco***: one token or two?
    - How do you decide it is one token?

Tokenization: Numbers

- Examples
  - ***3/12/91*** ***Mar. 12, 1991*** ***12/3/91***
  - ***55 B.C.***
  - ***B-52***
  - ***My PGP key is 324a3df234cb23e***
  - ***(800) 234-2333***
    - Often have embedded spaces
- Older IR systems may not index numbers
  - But often very useful
    - e.g., looking up error codes/stack traces on the web
- Will often index "meta-data" separately
  - Creation date, format, etc.

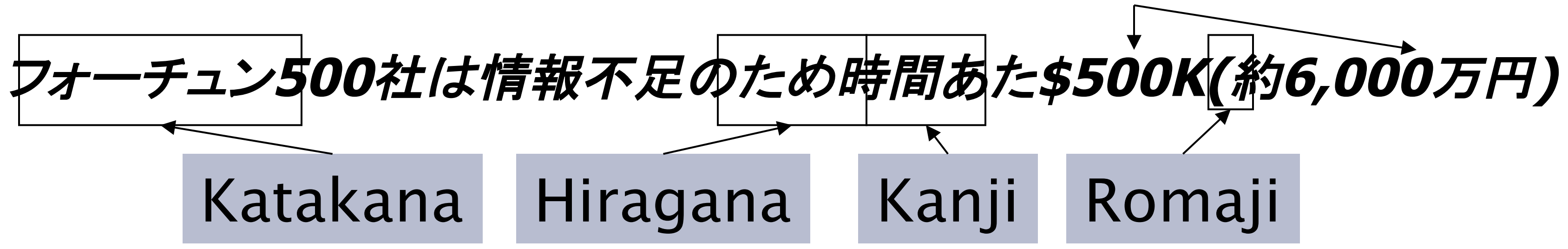
Tokenization: Language issues

- French
  - ***L'ensemble:*** one token or two?
    - ***L*** ? ***L'*** ? ***Le*** ?
- German noun compounds are not segmented
  - ***Lebensversicherungsgesellschaftsangestellter***
    - 'life insurance company employee'
  - German retrieval systems benefit greatly from a **compound splitter** module
    - Can give a 15% performance boost for German

Tokenization: Language issues

- Chinese and Japanese have no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - Not always guaranteed a unique tokenization

- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats



End-user can express query entirely in hiragana!

Stop words

- Stop list: exclude from dictionary the commonest words.
  - They have little semantic content: *'the', 'a', 'and', 'to', 'be'*
  - There are a lot of them: ~30% of postings for top 30 words
- But the trend is away from doing this:
  - Good compression techniques (IIR, Chapter 5)
    - the space for including stopwords in a system is very small
  - Good query optimization techniques (IIR, Chapter 7)
    - 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"

Normalization to terms

- Normalize words in indexed text (also query)
  - U.S.A.   USA
- Term is a (normalized) word type, which is an entry in our IR system dictionary
- We most commonly implicitly 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***
- Crucial: Need to “normalize” indexed text as well as query terms into the same form

Normalization to terms

- Do we handle synonyms and homonyms?
  - E.g., by hand-constructed equivalence classes
    - ***car*** = ***automobile*** ***color*** = ***colour***

- We can rewrite to form equivalence-class terms
  - When the doc contains ***automobile***, index it under ***car-automobile*** (and/or vice-versa)

Alternative to creating equivalence classe

- Or we can expand a query
  - When the query contains ***automobile***, look under ***car*** as well

Query expansion instead of normalization

- An alternative to equivalence classing is to do asymmetric expansion of query

- An example of where this may be useful
  - Enter: ***window*** Search: ***window, windows***
  - Enter: ***windows*** Search: ***Windows, windows***

- Potentially more powerful, but less efficient

Normalization: Case folding

- Reduce all letters to lower case
  - exception: upper case in mid-sentence?
    - e.g., ***General Motors***
    - ***Fed*** vs. ***fed***
    - ***SAIL*** vs. ***sail***
  - Often best to lower case everything, since users will use lowercase regardless of 'correct' capitalization…
- Google example: Query ***C.A.T.***
  - #1 result was for "cat" *not* Caterpillar Inc.

Normalization: Stemming and lemmatization

- For grammatical reasons, docs may use different forms of a word
  - Example: *organize*, *organizes*, and *organizing*

- There are families of derivationally related words with similar meanings
  - Example: *democracy*, *democratic*, and *democratization*

Lemmatization

- Reduce inflectional/variant forms to their base form, e.g.,
  - *am, are, is* $\rightarrow$ *be*
  - *car, cars, car's, cars'* $\rightarrow$ *car*
  - *the boy's cars are different colors* $\rightarrow$ *the boy car be different color*
- Lemmatization implies doing “proper” reduction to dictionary headword form
- It needs a complete vocabulary and morphological analysis to correctly lemmatize words

Stemming

- Reduce terms to their "roots" before indexing
  - Stemmers use language-specific rules, but they require less knowledge than a lemmatizer

- Stemming: crude affix chopping

- The exact stemmed form does not matter
  - only the resulted equivalence classes play role.

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

$\Rightarrow$

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

Porter's algorithm

- Commonest algorithm for stemming English
  - Results suggest it's at least as good as other stemming options

- Conventions + 5 phases of reductions
  - phases applied sequentially
  - each phase consists of a set of commands

Porter's algorithm: Typical rules

- *sses* → *ss*
- *ies* → *i*
- *ational* → *ate*
- *tional* → *tion*

- Rules sensitive to the *measure* of words
  - *(m>1) EMENT* →
    - *replacement* → *replac*
    - *cement* → *cement*

Do stemming and other normalizations help?

- English: very mixed results. Helps recall but harms precision
  - Example of harmful stemming:
    - operative (dentistry) ⇒ oper
    - operational (research) ⇒ oper
    - operating (systems) ⇒ oper

- Definitely useful for Spanish, German, Finnish, …
  - 30% performance gains for Finnish!

Lemmatization vs. Stemming

- Lemmatization produces <u>at most very modest benefits</u> for retrieval.

- Either form of normalization tends not to improve English information retrieval performance in aggregate

- The situation is different for languages with much more morphology (such as Spanish, German, and Finnish).
  - quite large gains from the use of stemmers

Normalization: Other languages

- Accents: e.g., French ***résumé*** vs. ***resume.***
- Umlauts: e.g., German: ***Tuebingen*** vs. ***Tübingen***
  - Should be equivalent

- Most important criterion:
  - How are your users like to write their queries for these words?
  - Users often may not type accents even in languages that standardly have accents
    - Often best to normalize to a de-accented term
      - ***Tuebingen, Tübingen, Tubingen*** → ***Tubingen***

- For foreign names, the spelling may be unclear or there may be variant transliteration standards giving different spellings
  - Soundex forms equivalence classes of words based on phonetic heuristics

Soundex

- Class of heuristics to expand a query into <u>phonetic equivalents</u>
  - Language specific (mainly for names)
  - E.g., ***chebyshev*** $\rightarrow$ ***tchebycheff***

- Invented for the U.S. census … in 1918

Soundex – typical algorithm

- Turn every token to be indexed into a 4-character reduced form
- Do the same with query terms
- Soundex index: Build and search an index on the reduced forms
  - Used when the query calls for a soundex match

http://www.creativyst.com/Doc/Articles/SoundEx1/SoundEx1.htm#Top

Soundex algorithm

1. Retain the first letter of the word.
2. Change all occurrences of the following letters to ‘0’:
   'A', E', 'I', 'O', 'U', 'H', 'W', 'Y' $\rightarrow$ 0
3. Change letters to digits as follows:
   - B, F, P, V $\rightarrow$ 1
   - C, G, J, K, Q, S, X, Z $\rightarrow$ 2
   - D,T $\rightarrow$ 3
   - L $\rightarrow$ 4
   - M, N $\rightarrow$ 5
   - R $\rightarrow$ 6

Soundex algorithm

4. Remove all pairs of consecutive digits.
5. Remove all zeros from the resulting string.
6. Pad the resulting string with trailing zeros and return the first four positions of the form:

   <uppercase letter> <digit> <digit> <digit>.

Example: ***Herman*** → H655.

Will ***hermann*** generate the same code?

Soundex

- Soundex is the classic algorithm
  - provided by most databases (Oracle, Microsoft, …)

- How useful is soundex?
  - Not very – for information retrieval
  - Okay for "high recall" tasks (e.g., Interpol), though biased to names of certain nationalities

- Zobel and Dart (1996) show that other algorithms for phonetic matching perform much better in the context of IR

Language-specificity

- Many of the above features embody transformations that are
  - Language-specific
  - Often, application-specific
- These are "plug-in" addenda to the indexing process
- Both open source and commercial plug-ins are available for handling these

Phrase and proximity queries: positional indexes

Phrase queries

- Example: "***stanford university***"
  - *"I went to university at Stanford"* is not a match.

- Easily understood by users
  - One of the few "advanced search" ideas that works
  - At least 10% of web queries are phrase queries
  - Many more queries are *implicit phrase queries*
    - such as person names entered without use of double quotes.

- It is not sufficient to store only the doc IDs in the posting lists

Approaches for phrase queries

- Indexing bi-words (two word phrases)
- Positional indexes
  - Full inverted index

Biword indexes

- Index every consecutive pair of terms in the text as a phrase
  - E.g., doc :"Friends, Romans, Countrymen"

    would generate these biwords:

    "friends romans" , "romans countrymen"

- Each of these biwords is now a dictionary term

- Two-word phrase query-processing is now immediate.

Biword indexes: Longer phrase queries

- Longer phrases are processed as conjunction of biwords

  Query: ***“stanford university palo alto”***

- can be broken into the Boolean query on biwords:

  “stanford university” *AND* ***“university palo”*** *AND* ***“palo alto”***

- Can have false positives!
  - Without the docs, we cannot verify that the docs matching the above Boolean query do contain the phrase.

Issues for biword indexes

- False positives (for phrases with more than two words)
- Index blowup due to bigger dictionary
  - Infeasible for more than biwords, big even for biwords
- Biword indexes are not the standard solution (for all biwords) but can be part of a compound strategy

Positional index

- In the postings, store for each ***term*** the position(s) in which tokens of it appear:

  <***term***, doc freq.;
  doc1: position1, position2 … ;
  doc2: position1, position2 … ; …>

<***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***"?

Positional index

- For phrase queries, we use a merge algorithm recursively at the doc level
- We need to deal with more than just equality of docIDs:
  - **Phrase query**: find places where all the words appear in sequence
  - **Proximity query**: to find places where all the words close enough

Processing a phrase query: Example

- Query: "***to be or not to be***"
- Extract inverted index entries for: ***to, be, or, not***
- Merge: find all positions of **"to"**, i, i+4, **"be"**, i+1, i+5, **"or"**, i+2, **"not"**, i+3.
  - ***to***:
    - <2:1,17,74,222,551>; <*4*:8,16,190,429,433, 512>; <7:13,23,191>; ...
  - ***be***:
    - <*1*:17,19>; <*4*:17,191,291,430,434>; <5:14,19,101>; ...
  - ***or***:
    - <3:5,15,19>; <*4*:5,100,251,431,438>; <7:17,52,121>; ...
  - ***not***:
    - <*4*:71,432>; <6:20,85>; ...

Positional index: Proximity queries

- k word proximity searches
  - Find places where the words are within k proximity
- Positional indexes can be used for such queries
  - as opposed to biword indexes
- Exercise: Adapt the linear merge of postings to handle proximity queries. Can you make it work for any value of *k*?

```
PositionalIntersect($p_1, p_2, k$)
 1  $answer \leftarrow \langle\ \rangle$
 2  while $p_1 \neq$ NIL and $p_2 \neq$ NIL
 3  do if $docID(p_1) = docID(p_2)$
 4        then $l \leftarrow \langle\ \rangle$
 5             $pp_1 \leftarrow positions(p_1)$
 6             $pp_2 \leftarrow positions(p_2)$
 7             while $pp_1 \neq$ NIL
 8             do while $pp_2 \neq$ NIL
 9                do if $|pos(pp_1) - pos(pp_2)| \leq k$
10                      then Add($l, pos(pp_2)$)
11                      else if $pos(pp_2) > pos(pp_1)$
12                              then break
13                   $pp_2 \leftarrow next(pp_2)$
14                while $l \neq \langle\ \rangle$ and $|l[0] - pos(pp_1)| > k$
15                do Delete($l[0]$)
16                for each $ps \in l$
17                do Add($answer, \langle docID(p_1), pos(pp_1), ps\rangle$)
18                $pp_1 \leftarrow next(pp_1)$
19             $p_1 \leftarrow next(p_1)$
20             $p_2 \leftarrow next(p_2)$
21        else if $docID(p_1) < docID(p_2)$
22               then $p_1 \leftarrow next(p_1)$
23               else $p_2 \leftarrow next(p_2)$
24  return $answer$
```

Positional index: size

- You can compress position values/offsets
  - Nevertheless, a positional index expands postings storage *substantially*
- Positional index is now standardly used
  - because of the power and usefulness of phrase and proximity queries …
  - used explicitly or implicitly in a ranking retrieval system.

Positional index: size

- Need an entry for each occurrence, not just once per doc
- Index size depends on average doc size
  - Average web page has <1000 terms
  - SEC filings, books, even some epic poems … easily 100,000 terms

Why?

- Consider a term with frequency 0.1%

Doc size (# of terms)	Expected Postings	Expected entries in Positional postings
1000	1	1
100,000	1	100

Positional index: size (rules of thumb)

- A positional index is usually 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

Phrase queries: Combination schemes

- Combining two approaches
  - For queries like ***"Michael Jordan",*** it is inefficient to merge positional postings lists
  - Good queries to include in the phrase index:
    - common queries based on recent querying behavior.
    - and also for phrases whose individual words are common but the phrase is not such common
      - Example: ***"The Who"***

Phrase queries: Combination schemes

- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme
  - needs (in average) ¼ of the time of using just a positional index
  - needs 26% more space than having a positional index alone

Resources

- IIR 2
- MIR 9.2
- Porter's stemmer:

  http://www.tartarus.org/~martin/PorterStemmer/