Information Retrieval

Text Analysis and Processing

Last lesson

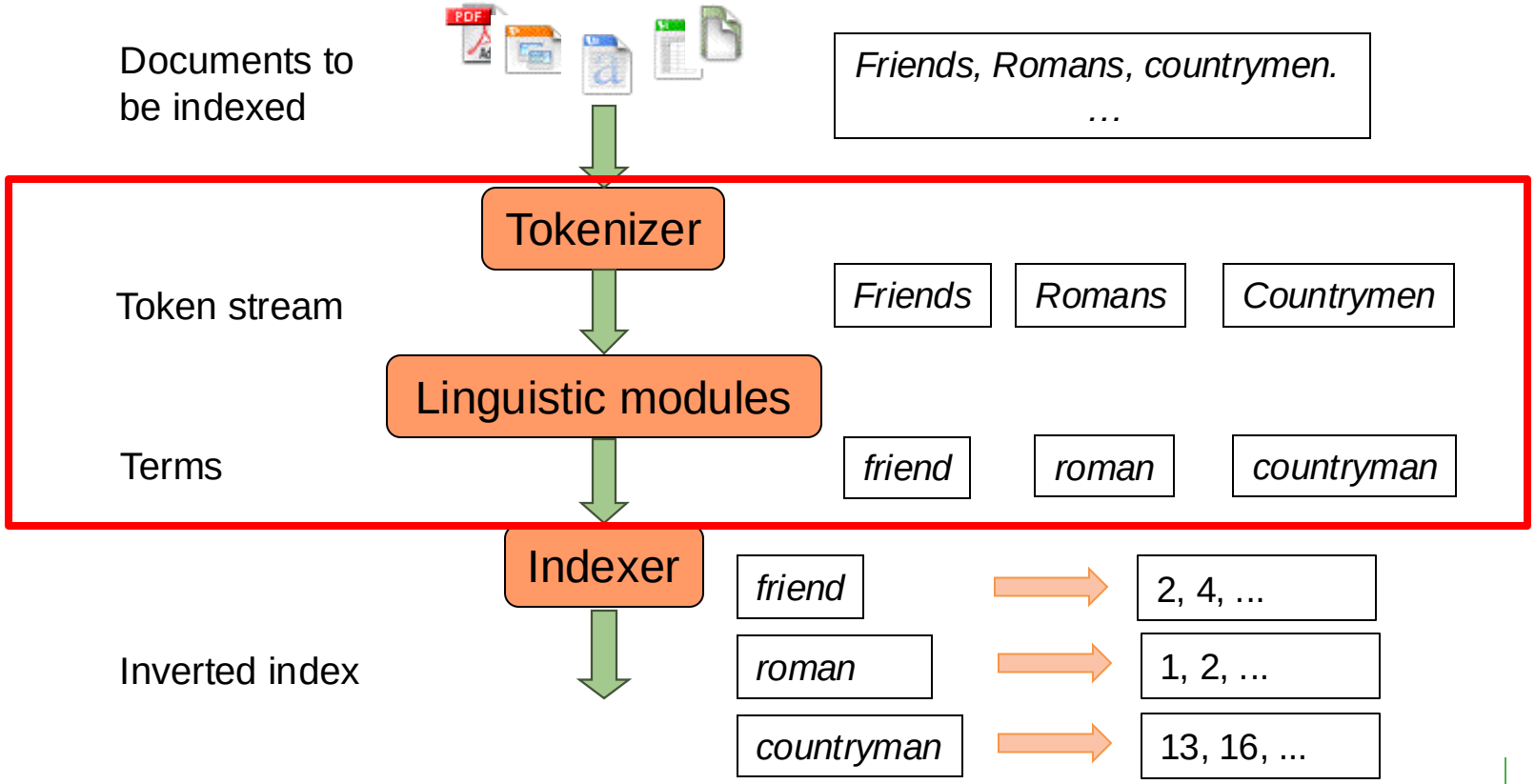
- Boolean indexing and search
- Term-document matrix
- Inverted index
- Inverted index construction

Plan for this lecture

- Preprocessing to form the term vocabulary
  - Tokenization
  - Stopping
  - Lemmatization
  - Stemming

- Handling phrases

Inverted index construction



Tokens and Terms

Tokenization

- Input: "Friends, Romans and Countrymen"
- Output: Tokens
  - Friends
  - Romans
  - Countrymen
- A token is an instance of a sequence of characters
- Each such token is now a candidate for an index entry, after further processing
  - Described below
- But what are valid tokens to emit?

Tokenization

- Alphanumeric sequences with 3 or more characters
- Example:
  - "Bigcorp's 2007 bi-annual report showed profits rose 10%."
- … becomes
  - "bigcorp 2007 annual report showed profits rose"
- Too simple for search applications or even large-scale experiments
- Why?
  - Information lost
  - Small decisions in tokenizing can have a major impact on the effectiveness of some queries

Tokenization

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

Numbers

- Examples
  - 3/20/91 Mar. 12, 1991 20/3/91
  - 55 B.C.
  - B-52
  - My PGP key is 324a3df234cb23e
  - (800) 234-2333
- Often have embedded spaces
- IR systems may not index numbers
  - But often very useful
    - think about things like looking up error codes/stacktraces on the web
  - (One answer is using n-grams: IIR ch. 3)
- Will often index "meta-data" separately
  - Creation date, format, etc.

Tokenization: language issues

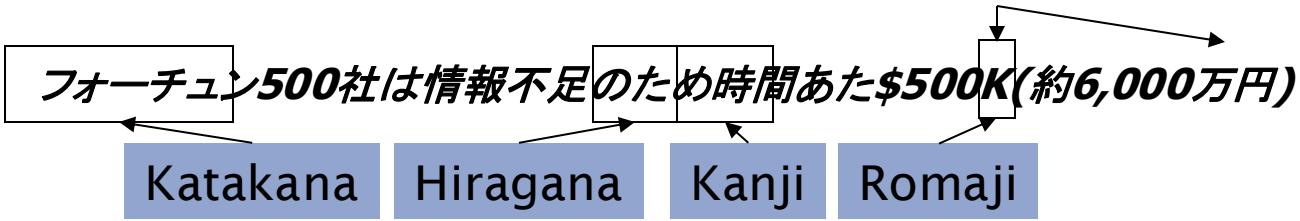
- French
  - L'ensemble → one token or two?
    - L ? L' ? Le ?
    - Want "l'ensemble" to match with "un ensemble"

- 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

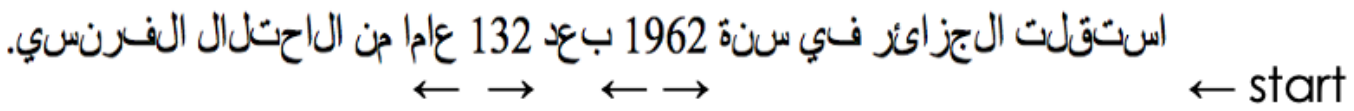
- Chinese and Japanese have no spaces between words:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - Not always guaranteed a unique tokenization
- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats



  - End-user can express query entirely in hiragana!

Tokenization: language issues

- Arabic (or Hebrew) is written right to left, but with certain items like numbers written left to right
- Words are separated, but letter forms within a word form complex ligatures
  - '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

Stop words

- With a stop list, you exclude from the dictionary entirely the commonest words. Intuition:
  - They have little semantic content: the, a, and, to, be
  - There are a lot of them: ~30% of postings for the top 30 words
- But the trend is away from doing this:
  - Good compression techniques mean the space for including stop words in a system is 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"

Normalization to terms

- We may need to "normalize" words in indexed text as well as query words into the same form
  - We want to match U.S.A. and USA
- Result is terms:
  - a 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

Normalization: other languages

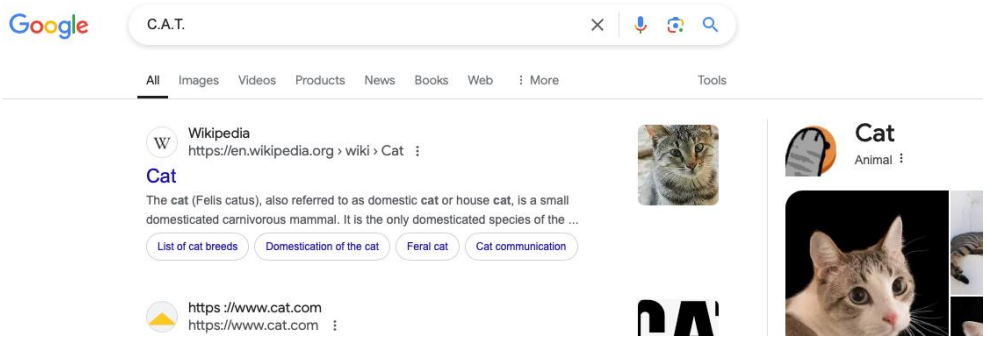
- Accents: e.g., French résumé vs. resume.
- Umlauts: e.g., German: Tuebingen vs. Tübingen
  - Should be equivalent
- Most important criterion:
  - How are your users like to write their queries for these words?

- Even in languages that standardly have accents, users often may not type them
  - Often best to normalize to a de-accented term
    - Tuebingen, Tübingen, Tubingen → Tubingen

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.



Thesauri and soundex

- 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 document contains automobile, index it under car-automobile (and vice-versa)
  - Or we can expand a query
    - When the query contains automobile, look under car as well
- What about spelling mistakes?
  - One approach is Soundex, which forms equivalence classes of words based on phonetic heuristics

Lemmatization

- Reduce inflectional/variant forms to base form
- E.g.,
  - am, are, is → be
  - car, cars, car's, cars' → car
- the boy's cars are different colors → the boy car be different color
- Lemmatization implies doing "proper" reduction to dictionary headword form

Stemming

- Many morphological variations of words
  - inflectional (plurals, tenses)
  - derivational (making verbs nouns etc.)
- In most cases, these have the same or very similar meanings
- Stemmers attempt to reduce morphological variations of words to a common stem
  - usually involves removing suffixes
- Can be done at indexing time or as part of query processing (like stopwords)

Stemming

- Two basic types
  - Dictionary-based: uses lists of related words
  - Algorithmic: uses program to determine related words
- Algorithmic stemmers
  - suffix-s: remove 's' endings assuming plural
    - e.g., cats → cat, lakes → lake, wiis → wii
  - Some problems:
    - supplies → supplie (should be supply)
    - ups → up (different concept)
- Some stemmers
  - Porter
  - Snowball
    - http://snowball.tartarus.org
  - Lancaster stemmer

Porter's algorithm

- Common 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
    - select the one that applies to the longest suffix.
- Typical rules
  - ational → ate (e.g., rational → rate)
  - tional → tion (e.g., conventional → convention)
  - sses → ss (e.g., guesses → guess)
  - ies → i (e.g., dictionaries → dictionari)
  - (m>1) EMENT →
    - replacement → replac
    - cement → cement

Language-specificity

- The above methods embody transformations that are
  - Language-specific, and often
  - Application-specific

- English: very mixed results. It helps recall for some queries but harms the precision on others
  - E.g., operative (dentistry) ⇒ oper
- Useful for Spanish, German, Finnish, …

Phrase queries and positional indexes

Phrase queries

- 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.
  - The concept of phrase queries has proven easily understood by users; one of the few "advanced search" ideas that works
  - Many more queries are implicit phrase queries
- For this, storing { term : [docs] } is not sufficient
  - Need more vocabulary's entries, OR
  - The posting list structure must be expanded

Phrases

- Text processing issue – how are phrases recognized?
- Two possible approaches:
  - Use word n-grams (n-words)
  - Store word positions in indexes and use proximity operators in queries

Phrases - 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 immediate.

Phrases - Biword indexes

- Longer phrases are processed by combining shorter phrases:
  - stanford university palo alto
- can be broken into the boolean query on biwords:
  - stanford university AND university palo AND palo alto
- Without the docs, we cannot verify that the docs matching the above Boolean query do contain the phrase.
  - we can have false positives!

- Index blowup due to bigger dictionary
  - Already big with biwords, infeasible for more than biwords
- Biword indexes are not the standard solution (for all biwords) but can be part of a compound strategy

Phrases - Positional indexes

- In the postings, store, for each term the position(s) in it appears:
  - <term, number of docs containing term;
  - doc1: position1, position2 … ;
  - doc2: position1, position2 … ;
  - etc.>
  - Note: to use the correct terminology, tokens appear in documents; the indexed form of a token is a term
    - Ex: "Operate an operating system..."
    - <oper; doc1: 0, 2>

Positional index example

- For phrase queries, we use a merge algorithm recursively at the document level
  - But we now need to deal with more than just equality

<**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

- Extract inverted index entries for each distinct term: to, be, or, not
- 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; ...
- Same general method for proximity searches

Positional index size

- Need an entry for each occurrence, not just once per document
- Index size depends on the average document size
  - Average web page has <1000 terms
  - Books, … easily 100,000 terms
- Consider a term with frequency 0.1%

Document size	Postings	Positional postings
1000	1	1
100,000	1	100

- A positional index is 2–4 as large as a non-positional index
- Positional index size 35–50% of the volume of the original text

This lesson

- Documents
- Tokenization
- Stopping
- Lemmatization
- Stemming
- Handling phrases: bigram index, positional index

Next lessons

- Index construction
  - Data structures
  - Indexing strategies
  - Distributed indexing
- Ranking
  - Weighting schemes
  - Vector space scoring