Preparing the Text for Indexing

Collecting the Documents

Documents to be indexed.

Tokenizer

Token stream.

Linguistic modules

Modified tokens.

Indexer

Inverted index.

Preparing the Text for Indexing

Collecting the Documents

Documents to be indexed.

Tokenizer

Token stream.

Linguistic modules

Modified tokens.

Indexer

Inverted index.

Complications: Format/language

What exactly is “a document”?

- What format is it in?
  - pdf/word/excel/html?
- What language is it in?
- What character set is in use?
- How easy is it to “read it in”?

Each of these is a classification problem, that can be tackled with machine learning.

But these tasks are often done heuristically …

Complications: Format/language

- What is a unit document?
  - An email? An email with 5 attachments?
  - A group of files (PPT or LaTeX in HTML)
- Documents being indexed can include documents from many different languages
  - An index may have to contain terms of several languages.
- Sometimes a document or its components can contain multiple languages/formats
  - French email with a German pdf attachment.
Words, Terms, Tokens, Types

- **Word** – A delimited string of characters as it appears in the text.
- **Term** – A "normalized" word (case, morphology, spelling etc); an equivalence class of words.
- **Token** – An instance of a word or term occurring in a document.
- **Type** – The same as a term in most cases; an equivalence class of tokens.

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?

What is a valid token?

- Finland’s capital → Finland? Finlands? Finland’s?
- Hewlett-Packard → Hewlett and Packard as two tokens?
- State-of-the-art → break up hyphenated sequence.
  - co-education → ?
- San Francisco: one token or two?
- San Francisco-Los Angeles highway: how many tokens?
- Dr. Summer address is 35 Winter St., 23014-1234, RI, USA.

Tokenization: Numbers

- 52 B.C.
- B-52
- My PGP key is 324a3df234cb23e
- 100.2.86.144

- Often, don’t index as text.
- But often very useful: e.g., for looking up error codes on the web.
- Often, we index “meta-data” separately:
  - Creation date, format, etc.
Tokenization: language issues

- East Asian languages (e.g., 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
- Semitic languages (Arabic, Hebrew) are written right to left, but certain items (e.g., numbers) written left to right.
- Words are separated, but letter forms within a word form complex ligatures

Linguistic Processing

- Normalization
- Capitalization/Case-folding
- Stop words
- Stemming
- Lemmatization

Linguistic Processing: Normalization

- Need to "normalize" terms in indexed text & in query terms into the same form
  - We want to match U.S.A. and USA
- We most commonly define equivalence classes of terms
  - e.g., by deleting periods in a term
- Alternative is to do asymmetric expansion:
  - Enter: window  Search: window, windows
  - Enter: windows  Search: Windows, windows
  - Enter: Windows  Search: Windows

Normalization: other languages

- Accents: résumé vs. resume.
- Most important criterion:
  - How are your users likely to write their queries for these words?
- Even in languages that standardly have accents, users often may not type them
- German: Tübingen vs. Tübingen
  - Should be equivalent
Linguistic Processing: 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...

Linguistic Processing: Stop Words

- With a stop list, you exclude from dictionary entirely the commonest words.
- Why do it:
  - They have little semantic content: the, a, and, to, be
  - They take a lot of space: ~30% of postings for top 30
    - You will measure this!
  - But the trend is away from doing this:
    - 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”

Linguistic Processing: Stemming

- Reduce terms to their “roots” before indexing
- “Stemming” suggest crude affix chopping
  - language dependent
  - e.g., automate(s), automatic, automation all reduced to automat.
- Porter’s Algorithm
  - Most common algorithm for stemming English
  - Results suggest at least as good as other stemming options
  - You find the algorithm and several implementations at http://tartarus.org/~martin/PorterStemmer/

Typical rules in Porter

- **Rule** | **Example**
  - sses → ss | caresses → caress
  - ies → i | butterflies → butterfly
  - ing → | meeting → meet
  - ional → tion | intentional → intention

- Weight of word sensitive rules
  - \((m>1)\) EMENT →
    - replacement → replac
    - cement → cement
An Example of Stemming

After introducing a generic search engine architecture, we examine each engine component in turn. We cover crawling, local Web page storage, indexing, and the use of link analysis for boosting search performance.

Linguistic Processing: Lemmatization

- Reduce inflectional/variant forms to base form
  - E.g.,
    - am, are, be, is → be
    - car, cars, car’s, cars’ → car

- Lemmatization implies doing “proper” reduction to dictionary headword form
  - the boy’s cars are different colors → the boy car be different color

Phrase queries

- Want 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 – they use quotes
  - about 10% of web queries are phrase queries
  - In average a query is 2.3 words long. (Is it still the case?)
- No longer suffices to store only <term : docs> entries

Phrase queries via Biword Indices

- Biwords: 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 (it works exactly like the one term process)
Longer phrase queries

* `stanford university palo alto` can be broken into the
  Boolean query on biwords:
  `stanford university AND university palo AND palo alto`

Without examining each of the returned docs,
we cannot verify that
the docs matching the above Boolean query
do contain the phrase.

Can have false positives!
(Why?)

Issues for biword indexes

* False positives, as noted before
* Index blowup due to bigger dictionary
* For extended biword index, parsing longer queries into conjunctions:
  * E.g., the query `tangerine trees, marmalade skies` is parsed into
    `tangerine trees AND trees marmalade AND marmalade skies`
* Not standard solution (for all biwords)

Better solution: Positional indexes

* Store, for each `term`, entries of the form:
  `<number of docs containing term;`
  `doc1: position1, position2 ... ;`
  `doc2: position1, position2 ... ;`
  `etc.>`
  `<be: 993427;`
  `1: 6 {7, 18, 33, 72, 86, 231};`
  `2: 2 {3, 149};`
  `4: 5 {17, 191, 291, 430, 434};`
  `5: 2 {363, 367};`
  ...>

Which of docs 1, 2, 4, 5 could contain "to be or not to be"?

Processing a phrase query

* Merge their `doc:position` lists to enumerate all positions
  with "to be or not to be".
  * to, 993427:
    * 2: 5{1,17,74,222,551};
    * 4: 5{8,16,190,429,433};
    * 7: 3{13,23,191}; ...
  * be, 178239:
    * 1: 2{17,19};
    * 4: 5{17,191,291,430,434};
    * 5: 3{14,19,101}; ...
* Same general method for proximity searches