

Information Retrieval and Organisation

Dell Zhang

Birkbeck, University of London

The Term Vocabulary and Postings Lists

Constructing Inverted Indexes

- ▶ The major steps in constructing an inverted index were:
 - ▶ Collect the documents to be indexed
 - ▶ Tokenize the text
 - ▶ Do linguistic preprocessing of tokens
 - ▶ Index the documents that each term occurs in
- ▶ How do we define and process the vocabulary of terms of a collection?

Obtaining Character Sequences

- ▶ Before we can even start worrying about terms, we need to deal with format and language of each document.
 - ▶ What format is it in? pdf, word, excel, html etc.
 - ▶ What language is it in?
 - ▶ What character set is in use?
- ▶ Each of these is a classification problem, which we will study later in this module
- ▶ For the moment assume that we know how to do this

Document Units

- ▶ So far we have assumed that documents are fixed units for the purposes of indexing
- ▶ Common approach: take each file in a folder as a document
- ▶ But that is not always the case:
 - ▶ Mbox e-mail format stores separate e-mails in one file
 - ▶ Archives such as zip, tar, or jar files contain many files
 - ▶ HTML may split up documents over many pages

Document Units

- ▶ For very long documents, the issue of indexing *granularity* arises
 - ▶ E.g. when indexing a collection of books, we could treat each book/chapter/paragraph as a document
 - ▶ If units get too small, we are likely to miss important information, as terms are distributed over different mini-documents
 - ▶ If units get too large, we tend to get spurious matches
- ▶ For making the right choice, a person deploying the system has to know the document collection, the users' information needs, and usage patterns.

Tokenization

- ▶ Tokenization is the task of chopping a character sequence into pieces, called *tokens*
- ▶ At the same time, certain characters might be eliminated (e.g. punctuation).
- ▶ Here's an example:

Input: Friends, Romans, Countrymen, lend me your ears;

Output:

Friends	Romans	Countrymen	lend	me	your	ears
---------	--------	------------	------	----	------	------

Tokenization

- ▶ Major question: what are the correct tokens to emit?
- ▶ Example on the previous slide looked fairly trivial:
 - ▶ Chop on whitespace
 - ▶ Throw away punctuation characters
- ▶ But this is only a starting point

Tokenization

- ▶ There are tricky cases (even in English)
 - ▶ Example: *Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.*
- ▶ Certain expressions shouldn't be split up:
 - ▶ C++, M*A*S*H, dell@dcs.bbk.ac.uk
- ▶ Splitting up hyphens:
 - ▶ co-education, Hewlett-Packard, hold-him-back-and-drag-him-away manoeuvre
- ▶ Splitting whitespace is not always clear:
 - ▶ York University, New York University
- ▶ Numbers are problematic:
 - ▶ 127.0.0.1, 15/01/2013

Problems with other languages

- ▶ Things get even more difficult in other languages
- ▶ Examples for missing whitespace:
 - ▶ Inuit:
tusaatsiarunnangittualuujunga
 - ▶ German:
Seitenfehlerunterbrechungsbehandlungsroutine
 - ▶ Chinese:
伯克贝克学院 (Birkbeck College) 是伦敦大学的一个学院，由 George Birkbeck 博士创立于 1823 年。1920 年与伦敦大学学院 (UCL)、伦敦政治经济学院 (LSE)、国王学院 (King's College)、伦敦商学院 (LBS) 等 31 机构共同组成伦敦大学。

Problems with other languages

▶ Example from Japanese:

ノーベル平和賞を受賞したワンガリ・マータイさんが名誉会長を務めるMOTTAINAIキャンペーンの一環として、毎日新聞社とマガジンハウスは「私の、もったいない」を募集します。皆様が日ごろ「もったいない」と感じて実践していることや、それにまつわるエピソードを800字以内の文章にまとめ、簡単な写真、イラスト、図などを添えて10月20日までにお送りください。大賞受賞者には、50万円相当の旅行券とエコ製品2点の副賞が贈られます。

- ▶ 4 different “alphabets”: Chinese characters, hiragana, katakana, latin (Western characters)
- ▶ No spaces (as in Chinese).
- ▶ End user can express query entirely in hiragana

Problems with other languages

- ▶ Sometimes you have to change direction while scanning text:

استقلت الجزائر في سنة 1962 بعد 132 عاما من الاحتلال الفرنسي.

← → ← → ← START

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

- ▶ Although, bidirectionality is not a problem if text is encoded in Unicode

Conclusion on Tokenization

- ▶ Trying to tackle all these problems is beyond the scope of this module (overlap with linguistics)
- ▶ We'll come back to some of the problems (for the English language) later in this course
 - ▶ Classification and categorization can help in solving some of these issues
- ▶ However, you should be aware of these problems

Normalization

- ▶ After document tokenization, do we just match query tokens to document token lists?
- ▶ Unfortunately, it's not as easy as this
- ▶ There are cases where tokens are not quite the same, but we still want to match them
 - ▶ Example: U.S.A. should match USA (or even US)

Normalization

- ▶ *Token normalization* is about transforming tokens into a standard form
- ▶ This allows matches to occur despite superficial differences
- ▶ Usual way to normalize is to create *equivalence classes*
 - ▶ For instance, the tokens *anti-discriminatory* and *antidiscriminatory* are both mapped onto the latter token
 - ▶ Searches for one term will retrieve documents that contain either

Normalization

- ▶ Alternative to equivalence classes are explicit rules (which may be asymmetric)
 - ▶ window \rightarrow window, windows
 - ▶ windows \rightarrow Windows, windows
 - ▶ Windows (no expansion)
- ▶ Writing explicit rules is quite powerful (but also quite costly to do)
- ▶ Some normalization may do more harm than good
 - ▶ Example: mapping C.A.T. to cat
- ▶ We will now look at some common techniques

Accents and Diacritics

- ▶ Have a fairly marginal status in English, so it's fairly safe to remove them
 - ▶ *cliché* vs. *cliche* or *naïve* vs. *naive*
- ▶ May be different for other languages
 - ▶ Spanish:
peña vs. *pena*
 - ▶ German: substitution of letters (for umlauts)
Universität vs. *Universitaet*

Case Folding

- ▶ Reduce all letters to lower case
- ▶ Possible exceptions: capitalized words in mid-sentence
 - ▶ *ETHICS* vs. *ethics*
 - ▶ *Fed* vs. *fed*
- ▶ It's often best to lowercase everything since users will use lowercase regardless of correct capitalization.

Lemmatization

- ▶ Reduce inflectional/variant forms to base form
 - ▶ Example: *am, are, is* → *be*
 - ▶ Example: *car, cars, car's, cars'* → *car*
 - ▶ Example: *the boy's cars are different colours* → *the boy car be different colour*
- ▶ Lemmatization implies doing “proper” reduction to dictionary headword form (the lemma).
 - ▶ Inflectional morphology (*cutting* → *cut*) vs. derivational morphology (*destruction* → *destroy*)

Stemming

- ▶ Stemming is defined as:
 - ▶ Crude heuristic process that chops off the ends of words
 - ▶ Trying to achieve what “principled” lemmatization attempts to do with a lot of linguistic knowledge
- ▶ Often inflectional **and** derivational
 - ▶ Example for derivational:
automate, automatic, automation all reduce to *automat*
- ▶ Language dependent
 - ▶ Fortunately, it works quite well for English

Porter Algorithm

- ▶ Most common algorithm for stemming English
 - ▶ Results suggest that it is at least as good as other stemming options
- ▶ Conventions + 5 phases of reductions
 - ▶ Phases are applied sequentially
 - ▶ Each phase consists of a set of commands.
- ▶ Sample command: *Delete the final 'ement' if what remains is longer than 1 character*
 - ▶ replacement → replac
 - ▶ cement → cement
- ▶ Sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix*

A Few Rules

Rule

SSES → SS

IES → I

SS → SS

S →

Example

caresses → caress

ponies → poni

caress → caress

cats → cat

Is Stemming Effective?

- ▶ In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.
 - ▶ Porter Stemmer equivalence class *oper* contains all of *operate operating operates operation operative operatives operational*.
 - ▶ Queries where stemming hurts:
 - “operational AND research” ,
 - “operating AND system” ,
 - “operative AND dentistry”

What Else Can We Do?

- ▶ After normalization, we have a list of *terms* that we can index
- ▶ However, there are some extremely common words which are of little value in helping select documents, e.g. “the”
- ▶ These words are called *stop words*
- ▶ Further examples: *a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with*

Stop Words

- ▶ We can eliminate stop words, i.e. we do not index them and eliminate them from queries
- ▶ Advantage: we save storage space, the postings lists of stop words tend to be very long
- ▶ Stop word elimination used to be standard in older IR systems.
- ▶ But you need stop words for phrase queries, e.g. “to be or not to be”
- ▶ Most web search engines index stop words.

Phrase Queries

- ▶ We want to answer a query such as “*stanford university*” – as a phrase.
- ▶ Thus “*The inventor Stanford Ovshinsky never went to university*” shouldn’t be a match.
- ▶ The concept of phrase query has proven easily understood by users.
- ▶ About 10% of web queries are phrase queries.
- ▶ Consequence for inverted index: no longer suffices to store docIDs in postings lists.
- ▶ Possible solutions?

Biword Index

- ▶ Index every consecutive pair of terms in the text as a phrase.
 - ▶ For example, *Friends, Romans, Countrymen* would generate two biwords: “*friends romans*” and “*romans countrymen*”
- ▶ Each of these biwords is now a vocabulary term.
- ▶ Two-word phrases can now easily be answered.

Biword Index

- ▶ However, there's a catch:
 - ▶ A document containing
“... *my friends, the Romans, have ... while
Romans and countrymen ...*”
would also be a match
- ▶ We need to do post-filtering of hits to identify subset that actually contains the 3-word phrase
- ▶ Biword indexes are rarely used
 - ▶ False positives, as noted above
 - ▶ Index blowup, due to very large term vocabulary

Positional Index

- ▶ Positional indexes are a more efficient alternative to biword indexes.
- ▶ Postings lists in a non-positional index: each posting is a docID only
- ▶ Postings lists in a positional index: each posting is a docID and a list of positions

Positional Index

▶ to_1 be_2 or_3 not_4 to_5 be_6

▶ to , 993427:

⟨ 1, 6: ⟨ 7, 18, 33, 72, 86, 231⟩;

2, 5: ⟨ 1, 17, 74, 222, 255⟩;

4, 5: ⟨ 8, 16, 190, 429, 433⟩;

5, 2: ⟨ 363, 367⟩;

7, 3: ⟨ 13, 23, 191⟩; ... ⟩

▶ be , 178239:

⟨ 1, 2: ⟨ 17, 25⟩;

4, 5: ⟨ 17, 191, 291, 430, 434⟩;

5, 3: ⟨ 14, 19, 101⟩; ... ⟩

Proximity Search

- ▶ We just saw how to use a positional index for phrase searches.
- ▶ We can also use it for proximity search.
 - ▶ For example: employment /3 place
 - ▶ Find all documents that contain employment and place within 3 words of each other.
 - ▶ *“Employment agencies that place healthcare workers are seeing growth”* is a hit.
 - ▶ *“Employment agencies that help place healthcare workers are seeing growth”* is not a hit.

Summary

- ▶ Before starting with the indexing, we should do some preprocessing of the documents
- ▶ We briefly sketched this preprocessing, as many techniques are outside of the scope of this module
- ▶ We will come back to some issues later