

PV211: Introduction to Information Retrieval

<https://www.fi.muni.cz/~sojka/PV211>

IIR 2: The term vocabulary and postings lists Handout version

Petr Sojka, Hinrich Schütze et al.

Faculty of Informatics, Masaryk University, Brno
Center for Information and Language Processing, University of Munich

2023-02-22

(compiled on 2023-02-15 08:37:49)

Overview

- 1 Recap
- 2 Documents
- 3 Terms
 - General + Non-English
 - English
- 4 Skip pointers
- 5 Phrase queries

Inverted index

For each term t , we store a list of all documents that contain t .

BRUTUS → 1 | 2 | 4 | 11 | 31 | 45 | 173 | 174

CAESAR → 1 | 2 | 4 | 5 | 6 | 16 | 57 | 132 | ...

CALPURNIA → 2 | 31 | 54 | 101

⋮


dictionary


postings

Westlaw: Example queries

Information need: Information on the legal theories involved in preventing the disclosure of trade secrets by employees formerly employed by a competing company

Query: "trade secret" /s disclos! /s prevent /s employe!

Information need: Requirements for disabled people to be able to access a workplace

Query: disab! /p access! /s work-site work-place (employment /3 place)

Information need: Cases about a host's responsibility for drunk guests

Query: host! /p (responsib! liab!) /p (intoxicat! drunk!) /p guest

Does Google use the Boolean model?

- On Google, the default interpretation of a query $[w_1 w_2 \dots w_n]$ is w_1 AND w_2 AND \dots AND w_n
- Cases where you get hits that do not contain one of the w_i :
 - anchor text
 - page contains variant of w_i (morphology, spelling correction, synonym)
 - long queries (n large)
 - boolean expression generates very few hits
- Simple Boolean vs. Ranking of result set
 - Simple Boolean retrieval returns matching documents in no particular order.
 - Google (and most well designed Boolean engines) rank the result set – they rank good hits (according to some estimator of relevance) higher than bad hits.

Take-away

- Understanding of the basic unit of classical information retrieval systems: **words** and **documents**: What is a document, what is a term?
- Tokenization: how to get from raw text to words (or tokens)
- More complex indexes: skip pointers and phrases

Major steps in inverted index construction

- 1 Collect the documents to be indexed.
- 2 Tokenize the text.
- 3 Do linguistic preprocessing of tokens.
- 4 Index the documents that each term occurs in.

Documents

- Last lecture: Simple Boolean retrieval system
- Our assumptions were:
 - We know what a document is.
 - We can “machine-read” each document.
- This can be complex in reality: “God is in the details.”
(Mies van der Rohe)

Parsing a document

- 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 course (IIR 13).
- Alternative: use heuristics

Format/Language: Complications

- A single index usually contains terms of several languages.
- Sometimes a document or its components contain multiple languages/formats.
 - French email with Spanish pdf attachment
- What is the document unit for indexing?
- A file?
- An email?
- An email with 5 attachments?
- A group of files (ppt or latex in HTML)?
- Upshot: Answering the question “what is a document?” is not trivial and requires some design decisions.
- Also: XML

Definitions

- **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 term in most cases: an equivalence class of tokens. More informally: what we consider same in the index, e.g. abstraction of a line in the incidence matrix.

Normalization

- Need to “normalize” words in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
- We most commonly implicitly define **equivalence classes** of terms.
- Alternatively: do asymmetric expansion
 - window → window, windows
 - windows → Windows, windows
 - Windows (no expansion)
- More powerful, but less efficient
- **Why don't you want to put *window*, *Window*, *windows*, and *Windows* in the same equivalence class?**

Normalization: Other languages

- Normalization and language detection interact.
- *PETER WILL NICHT MIT.* → MIT = mit
- *He got his PhD from MIT.* → MIT ≠ mit

Recall: Inverted index construction

- Input:

Friends, Romans, countrymen. So let it be with Caesar ...

- Output:

friend roman countryman so ...

- Each token is a candidate for a postings entry.
- What are valid tokens to emit?

Exercises

In June, the dog likes to chase the cat in the barn. – How many word tokens? How many word types?

Why tokenization is difficult – even in English. **Tokenize:** *Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.*

Tokenization problems: One word or two? (or several)

- Hewlett-Packard
- State-of-the-art
- co-education
- the hold-him-back-and-drag-him-away maneuver
- data base
- San Francisco
- Los Angeles-based company
- cheap San Francisco-Los Angeles fares
- York University vs. New York University

Numbers

- 22/02/22
- 2022-02-22
- Feb 22, 2022
- B-52
- 100.2.86.144
- (800) 234-2333
- 800.234.2333
- Older IR systems may not index numbers . . .
- . . . but generally it's a useful feature.
- Google example (1+1)

Chinese: No whitespace

莎拉波娃现在居住在美国东南部的佛罗里达。今年4月9日，莎拉波娃在美国第一大城市纽约度过了18岁生日。生日派对上，莎拉波娃露出了甜美的微笑。

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'.

Other cases of “no whitespace”

- Compounds in Dutch, German, Swedish, Czech (čistokapsonosoplena)
- Computerlinguistik → Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter
- → leben + versicherung + gesellschaft + angestellter
- Inuit: tusaatsiarunnangittualuujunga (I can't hear very well.)
- Many other languages with segmentation difficulties: Finnish, Urdu, . . .

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 syllabary for inflectional endings and function words, katakana syllabary for transcription of foreign words and other uses, and latin. No spaces (as in Chinese).

End user can express query entirely in hiragana!

Arabic script

ك ت ا ب ← كِتَابٌ

un b ā t i k

/kitābun/ ‘*a book*’

Arabic script: Bidirectionality

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

← → ← →

← START

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

Bidirectionality is not a problem if text is coded in Unicode.

Accents and diacritics

- Accents: résumé vs. resume (simple omission of accent)
- Umlauts: Universität vs. Universitaet (substitution with special letter sequence “ae”)
- Most important criterion: How are users likely to write their queries for these words?
- Even in languages that standardly have accents, users often do not type them. (Polish?)

Case folding

- Reduce all letters to lower case
- Even though case can be semantically meaningful
 - capitalized words in mid-sentence
 - MIT vs. mit
 - Fed vs. fed
 - ...
- It's often best to lowercase everything since users will use lowercase regardless of correct capitalization.

Stop words

- stop words = extremely common words which would appear to be of little value in helping to select documents matching a user need
- 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 word elimination used to be standard in older IR systems.
- But you need stop words for phrase queries, e.g. “King of Denmark”.
- Most web search engines index stop words.

More equivalence classing

- Soundex: IIR 3 (phonetic equivalence, Muller = Mueller)
- Thesauri: IIR 9 (semantic equivalence, car = automobile)

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 colors* → *the boy car be different color*
- Lemmatization implies doing “proper” reduction to dictionary headword form (the [lemma](#)).
- Inflectional morphology (*cutting* → *cut*) vs. derivational morphology (*destruction* → *destroy*)

Stemming

- Definition of stemming: Crude heuristic process that **chops off the ends of words** in the hope of achieving what “principled” lemmatization attempts to do with a lot of linguistic knowledge.
- Language dependent
- Often inflectional **and** derivational
- Example for derivational: *automate*, *automatic*, *automation* all reduce to *automat*

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 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.

Porter stemmer: A few rules

Rule

SSSES → SS

IES → I

SS → SS

S →

Example

caresses → caress

ponies → poni

caress → caress

cats → cat

Three stemmers: A comparison

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Porter stemmer: such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Lovins stemmer: such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Paice stemmer: such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Does stemming improve effectiveness?

- In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.
- Queries where stemming is likely to help: [tartan sweaters], [sightseeing tour san francisco]
- (equivalence classes: {sweater,sweaters}, {tour,tours})
- 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]

Exercise: What does Google do?

- Stop words
- Normalization
- Tokenization
- Lowercasing
- Stemming
- Non-latin alphabets
- Umlauts
- Compounds
- Numbers

Recall basic intersection algorithm

BRUTUS \rightarrow 1 \rightarrow 2 \rightarrow 4 \rightarrow 11 \rightarrow 31 \rightarrow 45 \rightarrow 173 \rightarrow 174

CALPURNIA \rightarrow 2 \rightarrow 31 \rightarrow 54 \rightarrow 101

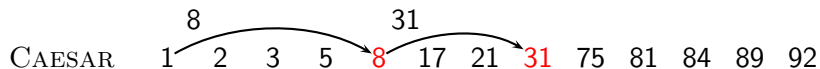
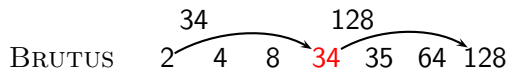
Intersection \Rightarrow 2 \rightarrow 31

- Linear in the length of the postings lists.
- Can we do better?

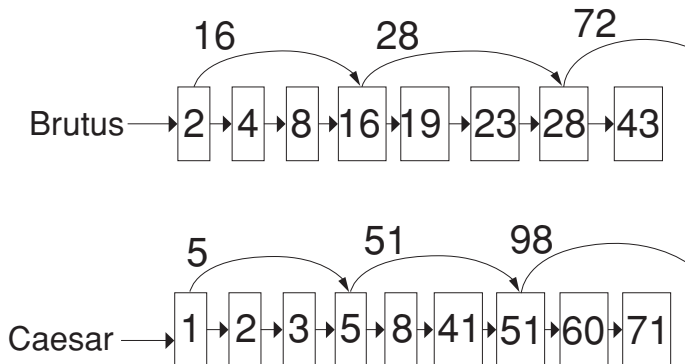
Skip pointers

- Skip pointers allow us to **skip** postings that will not figure in the search results.
- This makes intersecting postings lists more efficient.
- Some postings lists contain several million entries – so efficiency can be an issue even if basic intersection is linear.
- Where do we put skip pointers?
- How do we make sure intersection results are correct?

Basic idea



Skip lists: Larger example



Intersecting with skip pointers

```
INTERSECTWITHSKIPS( $p_1, p_2$ )
1  answer  $\leftarrow \langle \rangle$ 
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$ 
4      then  $\text{ADD}(\text{answer}, \text{docID}(p_1))$ 
5           $p_1 \leftarrow \text{next}(p_1)$ 
6           $p_2 \leftarrow \text{next}(p_2)$ 
7  else if  $\text{docID}(p_1) < \text{docID}(p_2)$ 
8      then if  $\text{hasSkip}(p_1)$  and  $(\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2))$ 
9          then while  $\text{hasSkip}(p_1)$  and  $(\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2))$ 
10             do  $p_1 \leftarrow \text{skip}(p_1)$ 
11             else  $p_1 \leftarrow \text{next}(p_1)$ 
12     else if  $\text{hasSkip}(p_2)$  and  $(\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1))$ 
13         then while  $\text{hasSkip}(p_2)$  and  $(\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1))$ 
14             do  $p_2 \leftarrow \text{skip}(p_2)$ 
15             else  $p_2 \leftarrow \text{next}(p_2)$ 
16 return answer
```

Where do we place skips?

- Tradeoff: number of items skipped vs. frequency skip can be taken
- More skips: Each skip pointer skips only a few items, but we can frequently use it.
- Fewer skips: Each skip pointer skips many items, but we can not use it very often.

Where do we place skips? (cont)

- Simple heuristic: for postings list of length P , use \sqrt{P} evenly-spaced skip pointers.
- This ignores the distribution of query terms.
- Easy if the index is static; harder in a dynamic environment because of updates.
- How much do skip pointers help?
- They used to help a lot.
- With today's fast CPUs, they don't help that much anymore.

Phrase queries

- We want to answer a query such as [Masaryk university] – as a phrase.
- Thus *The president Tomáš Garrigue Masaryk never went to Stanford university* should **not** 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: it no longer suffices to store docIDs in postings lists.
- Two ways of extending the inverted index:
 - biword index
 - positional index
 - **Any ideas?**

Biword indexes

- 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.

Longer phrase queries

- A long phrase like “*masaryk university brno*” can be represented as the Boolean query “MASARYK UNIVERSITY” AND “UNIVERSITY BRNO”
- We need to do post-filtering of hits to identify subset that actually contains the 3-word phrase.

Issues with biword indexes

- Why are biword indexes rarely used?
- False positives, as noted above
- Index blowup due to very large term vocabulary

Positional indexes

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

Positional indexes: Example

Query: "to₁ be₂ or₃ not₄ to₅ be₆"

TO, 993427:

- 1: ⟨7, 18, 33, 72, 86, 231⟩;
- 2: ⟨1, 17, 74, 222, 255⟩;
- 4: ⟨8, 16, 190, 429, 433⟩;
- 5: ⟨363, 367⟩;
- 7: ⟨13, 23, 191⟩; ...

BE, 178239:

- 1: ⟨17, 25⟩;
- 4: ⟨17, 191, 291, 430, 434⟩;
- 5: ⟨14, 19, 101⟩; ...

Document 4 is a match!

Exercise

Shown below is a portion of a positional index in the format: term: doc1: $\langle \text{position1, position2, } \dots \rangle$; doc2: $\langle \text{position1, position2, } \dots \rangle$; etc.

ANGELS: 2: $\langle 36,174,252,651 \rangle$; 4: $\langle 12,22,102,432 \rangle$; 7: $\langle 17 \rangle$;
FOOLS: 2: $\langle 1,17,74,222 \rangle$; 4: $\langle 8,78,108,458 \rangle$; 7: $\langle 3,13,23,193 \rangle$;
FEAR: 2: $\langle 87,704,722,901 \rangle$; 4: $\langle 13,43,113,433 \rangle$; 7: $\langle 18,328,528 \rangle$;
IN: 2: $\langle 3,37,76,444,851 \rangle$; 4: $\langle 10,20,110,470,500 \rangle$; 7: $\langle 5,15,25,195 \rangle$;
RUSH: 2: $\langle 2,66,194,321,702 \rangle$; 4: $\langle 9,69,149,429,569 \rangle$; 7: $\langle 4,14,404 \rangle$;
TO: 2: $\langle 47,86,234,999 \rangle$; 4: $\langle 14,24,774,944 \rangle$; 7: $\langle 19,319,599,709 \rangle$;
TREAD: 2: $\langle 57,94,333 \rangle$; 4: $\langle 15,35,155 \rangle$; 7: $\langle 20,320 \rangle$;
WHERE: 2: $\langle 67,124,393,1001 \rangle$; 4: $\langle 11,41,101,421,431 \rangle$; 7: $\langle 16,36,736 \rangle$;

Which document(s) if any match each of the following two queries, where each expression within quotes is a phrase query?: “fools rush in”, “fools rush in” AND “angels fear to tread”

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 /4 place`
- Find all documents that contain `EMPLOYMENT` and `PLACE` within 4 words of each other.
- *Employment agencies that place healthcare workers are seeing growth* is a hit.
- *Employment agencies that have learned to adapt now place healthcare workers* is not a hit.

Proximity search

- Use the positional index
- Simplest algorithm: look at cross-product of positions of (i) EMPLOYMENT in document and (ii) PLACE in document
- Very inefficient for frequent words, especially stop words
- Note that we want to return the actual matching positions, not just a list of documents.
- This is important for dynamic summaries, etc.

“Proximity” intersection

```

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

```

Combination scheme

- Biword indexes and positional indexes can be profitably combined.
- Many biwords are extremely frequent: Michael Jackson, Britney Spears, etc.
- For these biwords, increased speed compared to positional postings intersection is substantial.
- Combination scheme: Include frequent biwords as vocabulary terms in the index. Do all other phrases by positional intersection.
- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme. Faster than a positional index, at a cost of 26% more space for index.

“Positional” queries on Google

- For web search engines, positional queries are much more expensive than regular Boolean queries.
- Let's look at the example of phrase queries.
- Why are they more expensive than regular Boolean queries?
- Can you demonstrate on Google that phrase queries are more expensive than Boolean queries?

Take-away

- Understanding of the basic unit of classical information retrieval systems: **words** and **documents**: What is a document, what is a term?
- Tokenization: how to get from raw text to words (or tokens)
- More complex indexes: skip pointers and phrases

Resources

- Chapter 2 of IIR
- Resources at <https://www.fi.muni.cz/~sojka/PV211/> and <http://cislmu.org>, materials in MU IS and FI MU library
 - NLP IR boolean search system example: Sketch Engine: <https://ske.fi.muni.cz>
 - Google query language and preprocessing