NPFL103: Information Retrieval (1)

Introduction, Boolean retrieval, Inverted index, Text processing

Pavel Pecina

pecina@ufal.mff.cuni.cz

Institute of Formal and Applied Linguistics
Faculty of Mathematics and Physics
Charles University

Based on slides by Hinrich Schütze, University of Stuttgart.

Contents

Introduction

Definition of *Information Retrieval*

Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers).

Boolean retrieval

Boolean retrieval

- Boolean model is arguably the simplest model to base an information retrieval system on.
- Queries are Boolean expressions, e.g., CAESAR AND BRUTUS
- The search engine returns all documents that satisfy the Boolean expression.

Does Google use the Boolean model?

Does Google use the Boolean model?

- On Google, the default interpretation of a query $[w_1 \ w_2 \ldots w_n]$ is

$$w_1 \ \text{AND} \ w_2 \ \text{AND} \ \ldots \ \text{AND} \ w_n$$

- Cases where you get hits that do not contain one of the w_i:
  - anchor text (`<a href="http://web">anchor text</a>`)
  - page contains variant of w_i (morphology, spelling, synonymy)
  - long queries (n large)
  - boolean expression generates very few hits
- Other operators supported: NOT (-), OR (|), ...
- Simple Boolean vs. Ranking of result set
  - Simple Boolean retrieval returns documents in no particular order.
  - Google (and most well designed Boolean engines) rank the result set – good hits ranked higher than bad hits (according to some estimator of relevance).

Inverted index

Unstructured data in 1650: Plays of William Shakespeare

Unstructured data in 1650

- Which plays of Shakespeare contain the words Brutus and Caesar, but not Calpurnia?
- One could `grep` all of Shakespeare's plays for Brutus and Caesar, then strip out lines containing Calpurnia.
- Why is `grep` not the solution?
  - Slow (for large collections)
  - `grep` is line-oriented, IR is document-oriented
  - "not Calpurnia" is non-trivial
  - Other operations (e.g. search for Romans near country) infeasible

Term-document incidence matrix

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	1	1	0	0	0	1	
BRUTUS	1	1	0	1	0	0	
CAESAR	1	1	0	1	1	1	
CALPURNIA	0	1	0	0	0	0	
CLEOPATRA	1	0	0	0	0	0	
MERCY	1	0	1	1	1	1	
WORSER	1	0	1	1	1	0	
...							

Entry is 1 if term occurs. Example: CALPURNIA occurs in *Julius Caesar*.
Entry is 0 if term doesn't occur. Example: CALPURNIA doesn't occur in *The tempest*.

Incidence vectors

- So we have a 0/1 vector for each term.
- To answer the query Brutus AND Caesar AND NOT Calpurnia:
  1. Take the vectors for Brutus, Caesar, and Calpurnia
  2. Complement the vector of Calpurnia
  3. Do a (bitwise) AND on the three vectors:

     110100 AND 110111 AND 101111 = 100100

BRUTUS AND CAESAR AND NOT CALPURNIA

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
ANTHONY	1	1	0	0	0	1	
BRUTUS	1	1	0	1	0	0	
CAESAR	1	1	0	1	1	1	
CALPURNIA	0	1	0	0	0	0	
NOT CALPURNIA	1	0	1	1	1	1	
CLEOPATRA	1	0	0	0	0	0	
MERCY	1	0	1	1	1	1	
WORSER	1	0	1	1	1	0	
...							
result:	1	0	0	1	0	0	

Answers to query

Anthony and Cleopatra, Act III, Scene ii:

Agrippa [Aside to Domitius Enobarbus]: Why, Enobarbus,
When Antony found Julius Caesar dead,
He cried almost to roaring; and he wept
When at Philippi he found Brutus slain.

Hamlet, Act III, Scene ii:

Lord Polonius: I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.

Bigger collections

- Consider $N = 10^6$ documents, each with about 1000 tokens

  $\Rightarrow$ total of 10^9 tokens
- On average 6 bytes per token, including spaces and punctuation

  $\Rightarrow$ size of document collection is about $6 \cdot 10^9 = 6$ GB
- Assume there are $M = 500{,}000$ distinct terms in the collection

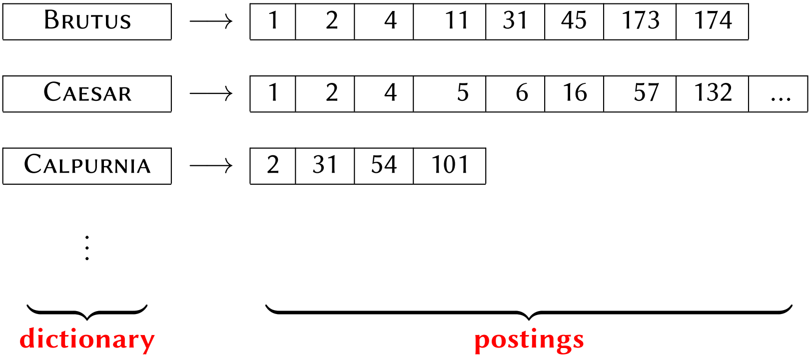
  $\Rightarrow M = 500{,}000 \times 10^6 =$ half a trillion 0s and 1s.
- But the matrix has no more than one billion 1s.

  $\Rightarrow$ Matrix is extremely sparse.
- What is a better representations?

  $\Rightarrow$ We only record the 1s.

Inverted Index

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



Inverted index construction

1. Collect the documents to be indexed:

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

2. Tokenize the text, turning each document into a list of tokens:

   | Friends | Romans | countrymen | So | ... |

3. Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms: | friend | roman | countryman | so | ...

4. Index the documents that each term occurs in by creating an inverted index, consisting of a dictionary and postings.

Tokenization and preprocessing

Doc 1. I did enact Julius Caesar: I was killed i' the Capitol; Brutus killed me.	⇒	**Doc 1.** i did enact julius caesar i was killed i' the capitol brutus killed me
Doc 2. So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:		**Doc 2.** so let it be with caesar the noble brutus hath told you caesar was ambitious

Generate postings, sort, create lists, determine document frequency

Doc 1. i did enact julius caesar i was killed i' the capitol brutus killed me

Doc 2. so let it be with caesar the noble brutus hath told you caesar was ambitious

⇒

term	docID
i	1
did	1
enact	1
julius	1
caesar	1
i	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

⇒

term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
i	1
i	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

⇒

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Split the result into dictionary and postings file

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 file**

Boolean queries

Simple conjunctive query (two terms)

- Consider the query: BRUTUS AND CALPURNIA
- To find all matching documents using inverted index:
  1. Locate BRUTUS in the dictionary
  2. Retrieve its postings list from the postings file
  3. Locate CALPURNIA in the dictionary
  4. Retrieve its postings list from the postings file
  5. Intersect the two postings lists
  6. Return intersection to user

Intersecting two postings lists

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

CALPURNIA ⟶ 2 → 31 → 54 → 101

Intersection ⟹ 2 → 31

- This is linear in the length of the postings lists.
- Note: This only works if postings lists are sorted.

Intersecting two postings lists

```
INTERSECT(p1, p2)
    answer ← ⟨ ⟩
    while p1 ≠ NIL and p2 ≠ NIL
    do if docID(p1) = docID(p2)
          then ADD(answer, docID(p1))
               p1 ← next(p1)
               p2 ← next(p2)
          else if docID(p1) < docID(p2)
                  then p1 ← next(p1)
                  else p2 ← next(p2)
    return answer
```

Boolean queries

- Boolean model can answer any query that is a Boolean expression.
  - Boolean queries use AND, OR and NOT to join query terms.
  - Views each document as a set of terms.
  - Is precise: Document matches condition or not.
- Primary commercial retrieval tool for 3 decades
- Many professional searchers (e.g., lawyers) still like Boolean queries.
  - You know exactly what you are getting.

Text processing

Documents

- So far: Simple Boolean retrieval system
- Our assumptions were:
  1. We know what a document is.
  2. We can “machine-read” each document.
- This can be complex in reality.

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 (see later)
- 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 (e.g. 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.

Definitions

- Word – A delimited string of characters as it appears in the text.
- Term – A “normalized” word (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.

Normalization

- Need to "normalize" terms 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* $\rightarrow$ *window, windows*
  - *windows* $\rightarrow$ *Windows, windows*
  - *Windows* $\rightarrow$ *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.
- Example:
  - *PETER WILL NICHT MIT.* $\rightarrow$ MIT $=$ mit
  - *He got his PhD from MIT.* $\rightarrow$ MIT $\neq$ 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

- How many word tokens? How many word types?

  Example 1: *In June, the dog likes to chase the cat in the barn.*

  Example 2: *Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.*

- ...tokenization is difficult – even in English.

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

- 3/20/91
- 20/3/91
- Mar 20, 1991
- 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.

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

和尚

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
- Computerlinguistik $\rightarrow$ Computer + Linguistik
- Lebensversicherungsgesellschaftsangestellter
- $\rightarrow$ leben + versicherung + gesellschaft + angestellter
- Inuit: tusaatsiarunnanngittualuujunga (I can't hear very well.)
- 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
  - Latin
- No spaces (as in Chinese).
- End user can express query entirely in hiragana!

Arabic script

كِتَابٌ ⇐ ك ِ ت ا ب ٌ

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 “ä” and “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 (e.g. Czech)

Case folding

- Reduce all letters to lower case
- Possible exceptions: capitalized words in mid-sentence

  Example: 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 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: phonetic equivalence, e.g. *Muller* = *Mueller*
- Thesauri: semantic equivalence, e.g. *car* = *automobile*

Lemmatization

- Reduce inflectional/variant forms to base form
- Examples:
  - *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 (the lemma).
- Two types:
  - inflectional (*cutting* $\rightarrow$ *cut*)
  - derivational (*destruction* $\rightarrow$ *destroy*)

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 (derivational): *automate, automatic, automation* all reduce to *automat*

Porter algorithm (1980)

- Most common algorithm for stemming English
- Results suggest that it is at least as good as other stemming options (1980!)
- Conventions + 5 phases of reductions 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			**Example**		
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
SS	→	SS	caress	→	caress
S	→		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 analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

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}*
- Queries where stemming hurts:
  - [OPERATIONAL RESEARCH], [OPERATING SYSTEM], [OPERATIVE DENTISTRY]
  - Porter Stemmer equivalence class *oper* contains all of *operate, operating, operates, operation, operative, operatives, operational.*

Phrase queries

Phrase queries

- We answer a query such as [STANFORD UNIVERSITY] – as a phrase.
- Thus *The inventor Stanford Ovshinsky never went to 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:
  1. biword index
  2. positional index

Biword indexes

- Index every consecutive pair of terms in the text as a phrase.
- Example: *Friends, Romans, Countrymen* 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 *"stanford university palo alto"* can be represented as the Boolean query "STANFORD UNIVERSITY" AND "UNIVERSITY PALO" AND "PALO ALTO"
- We need to do post-filtering of hits to identify subset that actually contains the 4-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: "to_1 be_2 or_3 not_4 to_5 be_6"

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!

Proximity search

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 all combinations 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(p1, p2, k)
answer ← ⟨ ⟩
while p1 ≠ NIL and p2 ≠ NIL
do if docID(p1) = docID(p2)
      then l ← ⟨ ⟩
           pp1 ← positions(p1)
           pp2 ← positions(p2)
           while pp1 ≠ NIL
           do while pp2 ≠ NIL
                 do if |pos(pp1) − pos(pp2)| ≤ k
                       then Add(l, pos(pp2))
                       else  if pos(pp2) > pos(pp1)
                                then break
                    pp2 ← next(pp2)
                 while l ≠ ⟨ ⟩ and |l[0] − pos(pp1)| > k
                 do Delete(l[0])
                 for  each ps ∈ l
                 do Add(answer, ⟨docID(p1), pos(pp1), ps⟩)
                 pp1 ← next(pp1)
           p1 ← next(p1)
           p2 ← next(p2)
      else  if docID(p1) < docID(p2)
               then p1 ← next(p1)
               else  p2 ← next(p2)
return answer
```

Combination scheme

- Biword indexes and positional indexes can be profitably combined.
- Many biwords extremely frequent: *Michael Jackson, Lady Gaga* 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?