Modern Information Retrieval

Boolean information retrieval and document preprocessing[1]

Hamid Beigy

Sharif university of technology

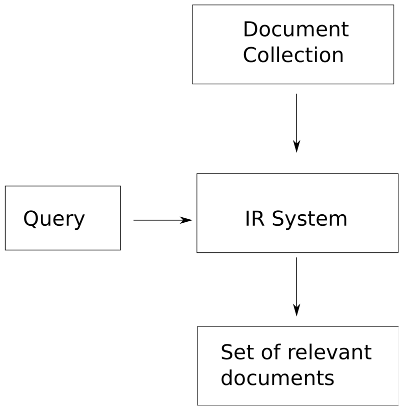
September 30, 2023

[1]Some slides have been adapted from slides of Manning, Yannakoudakis, and Schütze.

Table of contents

Introduction

Introduction



1. Document Collection: units we have built an IR system over.
2. An information need is the topic about which the user desires to know more about.
3. A query is what the user conveys to the computer in an attempt to communicate the information need.

Boolean Retrieval Model

Boolean Retrieval Model

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

Unstructured data in 1650

1. Which plays of Shakespeare contain the words BRUTUS AND CAESAR, but NOT CALPURNIA?
2. One could grep all of Shakespeare's plays for BRUTUS and CAESAR, then strip out lines containing CALPURNIA.
3. Why is grep not the solution?
   - 3.1 Slow (for large collections)
   - 3.2 grep is line-oriented, IR is document-oriented
   - 3.3 NOT CALPURNIA is non-trivial
   - 3.4 Other operations (e.g., find the word ROMANS near COUNTRYMAN) not feasible

Term-document incidence matrix

Example

	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

1. So we have a 0/1 vector for each term.
2. To answer the query BRUTUS AND CAESAR AND NOT CALPURNIA:
   2.1 Take the vectors for BRUTUS, CAESAR, and CALPURNIA
   2.2 Complement the vector of CALPURNIA
   2.3 Do a (bitwise) AND on the three vectors
   2.4 110100 AND 110111 AND 101111 = 100100

0/1 vectors and result of bitwise operations

Example

	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	
. . .							
result:	1	0	0	1	0	0	

The results are two documents

Antony and Cleopatra, Act III, Scene ii

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

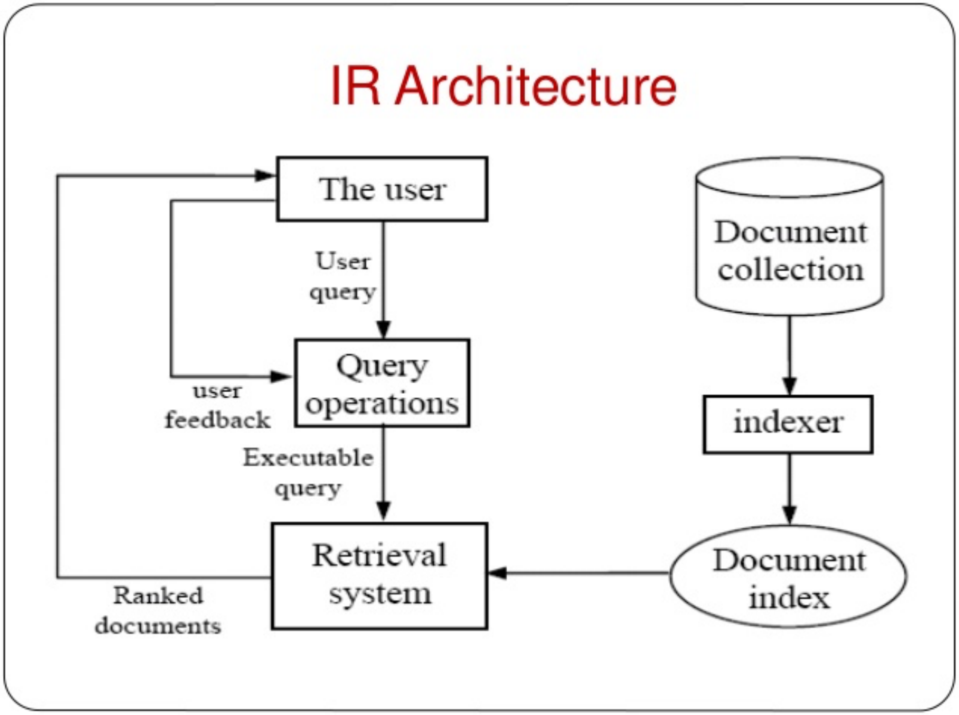
Hamlet, Act III, Scene ii

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

Bigger collections

1. Consider $N = 10^6$ documents, each with about 1000 tokens $\Rightarrow$ total of 10^9 tokens
2. On average 6 bytes per token, including spaces and punctuation $\Rightarrow$ size of document collection is about $6 \times 10^9 = 6\ GB$
3. Assume there are $M = 500{,}000$ distinct terms in the collection
4. $M = 500{,}000 \times 10^6 =$ half a trillion 0s and 1s.
5. But the matrix has no more than one billion 1s.
   5.1 Matrix is extremely sparse.
6. What is a better representations?
   6.1 We only record the 1s.

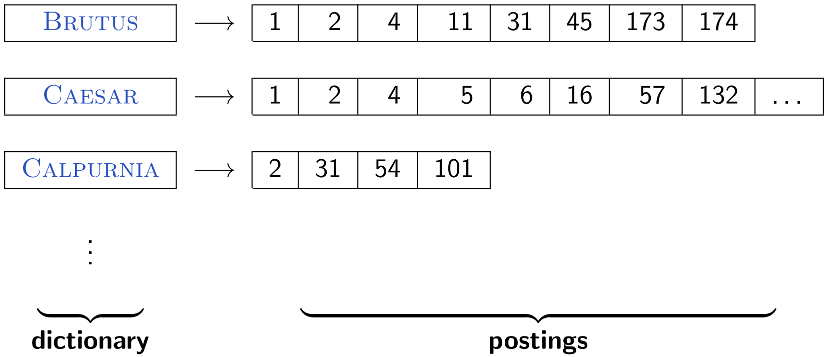
Architecture of IR systems



Inverted index

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

Inverted index construction

1. Collect the documents to be indexed:
   $\boxed{\text{Friends, Romans, countrymen.}}$ $\boxed{\text{So let it be with Caesar}}$ ...
2. Tokenize the text, turning each document into a list of tokens:
   $\boxed{\text{Friends}}$ $\boxed{\text{Romans}}$ $\boxed{\text{countrymen}}$ $\boxed{\text{So}}$ ...
3. Do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms: $\boxed{\text{friend}}$ $\boxed{\text{roman}}$ $\boxed{\text{countryman}}$ $\boxed{\text{so}}$ ...
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 2. So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:

$\Longrightarrow$

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

Example: index creation by sorting

Doc 1:
I did enact Julius Caesar: I was killed i' the Capitol;Brutus killed me.

$\Longrightarrow$ Tokenisation

Doc 2:
So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious.

$\Longrightarrow$ Tokenisation

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

$\Longrightarrow$ Sorting

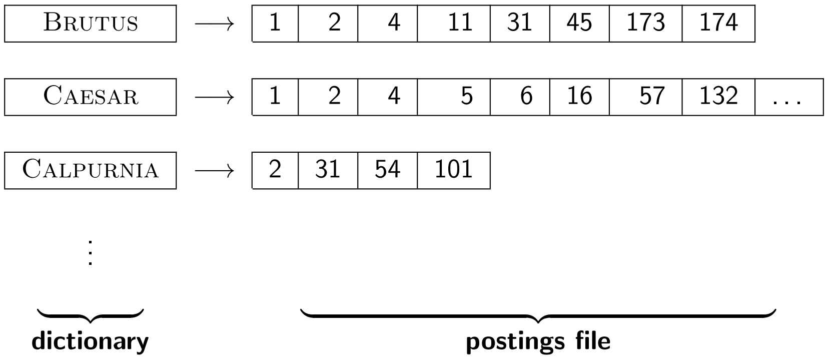
Term (sorted)	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	2
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	2
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	1
with	2

Index creation (grouping step)

Term	doc. freq.		Postings list
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

1. Primary sort by term (dictionary)
2. Secondary sort (within postings list) by document ID
3. Document frequency (= length of postings list):
   3.1 for more efficient Boolean searching (we discuss later)
   3.2 for term weighting (we discuss later)
4. Keep Dictionary in memory
5. Postings List (much larger) traditionally on disk

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

Processing Boolean queries

Simple conjunctive query (two terms)

1. Consider the query: Brutus AND Calpurnia
2. To find all matching documents using inverted index:
   2.1 Locate Brutus in the dictionary
   2.2 Retrieve its postings list from the postings file
   2.3 Locate Calpurnia in the dictionary
   2.4 Retrieve its postings list from the postings file
   2.5 Intersect the two postings lists
   2.6 Return intersection to user

Intersecting two postings lists

$$\text{B}\textsc{rutus} \longrightarrow \boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow \boxed{11} \rightarrow \boxed{31} \rightarrow \boxed{45} \rightarrow \boxed{173} \rightarrow \boxed{174}$$

$$\text{C}\textsc{alpurnia} \longrightarrow \boxed{2} \rightarrow \boxed{31} \rightarrow \boxed{54} \rightarrow \boxed{101}$$

$$\text{Intersection} \Longrightarrow \boxed{2} \rightarrow \boxed{31}$$

1. This is linear in the length of the postings lists.
2. 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)
        if docID(p1) < docID(p2)
             then p1← next(p1)
             else p2← next(p2)
   return answer
```

$$\begin{tikzcd}
\boxed{\text{Brutus}} \arrow[r] & \boxed{1} \arrow[r] & \boxed{2} \arrow[r] & \boxed{4} \arrow[r] & \boxed{11} \arrow[r] & \boxed{31} \arrow[r] & \boxed{45} \arrow[r] & \boxed{173} \arrow[r] & \boxed{174} \\
\boxed{\text{Calpurnia}} \arrow[r] & \boxed{2} \arrow[r] & \boxed{31} \arrow[r] & \boxed{54} \arrow[r] & \boxed{101}
\end{tikzcd}$$

Intersection $\boxed{2}$ $\boxed{31}$

Complexity of the Intersection Algorithm

1. Bounded by worst-case length of postings lists
2. Thus, formally, querying complexity is $O(N)$, with N the number of documents in the document collection
3. But in practice, much better than linear scanning, which is asymptotically also $O(N)$.

Query processing: Exercise

$$\begin{tikzcd}[column sep=small]
\text{FRANCE} \arrow[r] & \boxed{1} \arrow[r] & \boxed{2} \arrow[r] & \boxed{3} \arrow[r] & \boxed{4} \arrow[r] & \boxed{5} \arrow[r] & \boxed{7} \arrow[r] & \boxed{8} \arrow[r] & \boxed{9} \arrow[r] & \boxed{11} \arrow[r] & \boxed{12} \arrow[r] & \boxed{13} \arrow[r] & \boxed{14} \arrow[r] & \boxed{15} \\
\text{PARIS} \arrow[r] & \boxed{2} \arrow[r] & \boxed{6} \arrow[r] & \boxed{10} \arrow[r] & \boxed{12} \arrow[r] & \boxed{14} \\
\text{LEAR} \arrow[r] & \boxed{12} \arrow[r] & \boxed{15}
\end{tikzcd}$$

Compute hit list for ((PARIS AND NOT FRANCE) OR LEAR)

Boolean retrieval model: Assessment

1. The Boolean retrieval model can answer any query that is a Boolean expression.
   1.1 Boolean queries are queries that use AND, OR and NOT to join query terms.
   1.2 Views each document as a set of terms.
   1.3 Is precise: Document matches condition or not.
2. Primary commercial retrieval tool for 3 decades
3. Many professional searchers (e.g., lawyers) still like Boolean queries.
   3.1 You know exactly what you are getting.
4. Many search systems you use are also Boolean: spotlight, email, intranet etc.

Commercially successful Boolean retrieval: Westlaw

1. Largest commercial legal search service in terms of the number of paying subscribers
2. Over half a million subscribers performing millions of searches a day over tens of terabytes of text data
3. The service was started in 1975.
4. In 2005, Boolean search (called “Terms and Connectors” by Westlaw) was still the default, and used by a large percentage of users . . .
5. . . . although ranked retrieval has been available since 1992.

Does Google use the Boolean model?

1. On Google, the default interpretation of a query $[w_1\ w_2\ \ldots w_n]$ is w_1 AND w_2 AND $\ldots$ AND w_n
2. Cases where you get hits that do not contain one of the w_i:
    2.1 anchor text
    2.2 page contains variant of w_i (morphology, spelling correction, synonym)
    2.3 long queries (n large)
    2.4 boolean expression generates very few hits
3. Simple Boolean vs. Ranking of result set
    3.1 Simple Boolean retrieval returns matching documents in no particular order.
    3.2 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.

Optimization

Query optimization

1. Example query: Brutus AND Calpurnia AND Caesar
2. Simple and effective optimization: Process in order of increasing frequency
3. Start with the shortest postings list, then keep cutting further
4. In this example, first Caesar, then Calpurnia, then Brutus

$$\text{Brutus} \longrightarrow \boxed{1}\to\boxed{2}\to\boxed{4}\to\boxed{11}\to\boxed{31}\to\boxed{45}\to\boxed{173}\to\boxed{174}$$

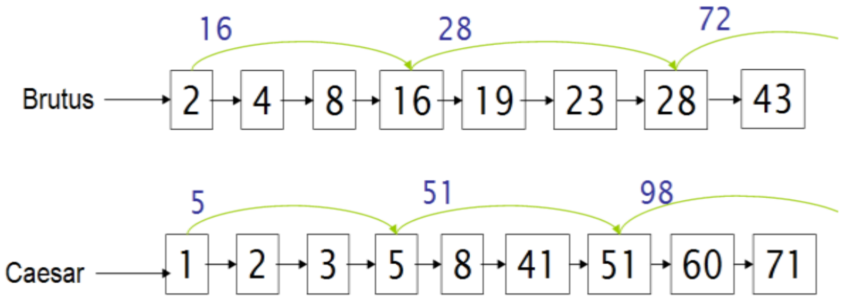
$$\text{Calpurnia} \longrightarrow \boxed{2}\to\boxed{31}\to\boxed{54}\to\boxed{101}$$

$$\text{Caesar} \longrightarrow \boxed{5}\to\boxed{31}$$

Optimized intersection algorithm for conjunctive queries

```
INTERSECT($\langle t_1, \ldots, t_n \rangle$)
1  $terms \leftarrow$ SORTBYINCREASINGFREQUENCY($\langle t_1, \ldots, t_n \rangle$)
2  $result \leftarrow postings(first(terms))$
3  $terms \leftarrow rest(terms)$
4  while $terms \neq$ NIL and $result \neq$ NIL
5  do $result \leftarrow$ INTERSECT($result, postings(first(terms))$)
6      $terms \leftarrow rest(terms)$
7  return $result$
```

Skip lists

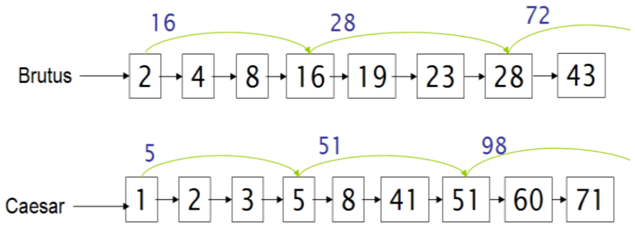


1. Augment postings lists with skip pointers (at indexing time)
2. If skip-list pointer present, skip multiple entries
   Example: after we match $8, 16 < 41$, skip to item after skip pointer
3. How many skip-list pointers do we use?
   Heuristic: for postings lists of length L, use $\sqrt{L}$ evenly-spaced skip pointers

Intersection with skip pointers

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

Where do we place skips?



1. Number of items skipped vs. frequency that skip can be taken
   1.1 More skips: each pointer skips only a few items, but we can frequently use it, but many comparisons.
   1.2 Fewer skips: each skip pointer skips many items, but we can not use it very often, but fewer comparisons.
2. This ignores the distribution of query terms.
3. Easy for static index; hard in dynamic environments due to updates.
4. How much do skip pointers help? They used to help a lot.
5. With today's fast CPUs, they don't help that much anymore.

Phrase Queries

1. We want to answer a query such as STANFORD UNIVERSITY as a phrase.
2. THE INVENTOR STANFORD OVSHINSKY NEVER WENT TO UNIVERSITY should not be a match.
3. The concept of phrase query has proven easily understood by users.
4. About 10% of web queries are phrase queries (double-quotes syntax).
5. Consequence for inverted indexes: no longer sufficient to store docIDs in postings lists.
6. Two ways of extending the inverted index:
   6.1 biword index
   6.2 positional index

Biword index

1. Index every consecutive pair of terms in the text as a phrase

Example

For document: FRIENDS, ROMANS, COUNTRYMEN

Generate two following biwords

FRIENDS ROMANS and ROMANS COUNTRYMEN

2. Each of these biwords is now a dictionary term.
3. Two-word phrases can now easily be answered.
4. A long phrase like STANFORD UNIVERSITY PALO ALTO can be broken into the Boolean query
   STANFORD UNIVERSITY AND UNIVERSITY PALO AND PALO ALTO
5. False positives. we need to do post-filtering of hits to identify subset that actually contains the 4-word phrase.

Issues with biword index

1. Why is biword index rarely used?
2. False positives, as noted above
3. Index blowup due to very large dictionary / vocabulary
   3.1 Searches for a single term?
   3.2 Infeasible for more than bigrams

Positional indexes

1. Positional indexes are a more efficient alternative to biword indexes.
2. Postings lists in a nonpositional index: each posting is just a docID
3. Postings lists in a positional index: each posting is a docID and a list of positions (offsets).

Positional indexes

1. Query: TO BE OR NOT TO BE

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>;
. . .	. . . >

2. Document 4 matches. Why? (Always: term, doc freq, docid, offsets)

Proximity search

1. We just saw how to use a positional index for phrase searches.
2. We can also use it for proximity search.
3. Example: EMPLOYMENT /4 PLACE
4. 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.
5. Note that we want to return the actual matching positions, not just a list of documents.
6. Use the positional index

Proximity intersection

```
$\textsc{PositionalIntersect}(p_1, p_2, k)$
answer $\leftarrow \langle\, \rangle$
while $p_1 \neq$ NIL and $p_2 \neq$ NIL
do if $docID(p_1) = docID(p_2)$
      then $l \leftarrow \langle\, \rangle$
           $pp_1 \leftarrow positions(p_1)$
           $pp_2 \leftarrow positions(p_2)$
           while $pp_1 \neq$ NIL
           do while $pp_2 \neq$ NIL
              do if $|pos(pp_1) - pos(pp_2)| \leq k$
                    then $\textsc{Add}(l, pos(pp_2))$
                    else  if $pos(pp_2) > pos(pp_1)$
                            then break
                 $pp_2 \leftarrow next(pp_2)$
              while $l \neq \langle\, \rangle$ and $|l[0] - pos(pp_1)| > k$
              do $\textsc{Delete}(l[0])$
              for  each $ps \in l$
              do $\textsc{Add}(answer, \langle docID(p_1), pos(pp_1), ps \rangle)$
              $pp_1 \leftarrow next(pp_1)$
           $p_1 \leftarrow next(p_1)$
           $p_2 \leftarrow next(p_2)$
      else  if $docID(p_1) < docID(p_2)$
              then $p_1 \leftarrow next(p_1)$
              else  $p_2 \leftarrow next(p_2)$
return answer
```

Combination scheme

1. Biword indexes and positional indexes can be profitably combined.
2. Many biwords are extremely frequent.
3. For frequent biwords, increased speed compared to positional postings intersection is substantial.
4. Combination scheme: Include frequent biwords as vocabulary terms in the index. Do all other phrases by positional intersection.

More general optimization

1. Example query: (MADDING OR CROWD) AND (IGNOBLE OR STRIFE)
2. Get frequencies for all terms
3. Estimate the size of each OR by the sum of its frequencies (conservative)
4. Process in an increasing order of OR sizes

Document preprocessing

Documents

1. Up to now, to build an inverted index, we assumed that
   1.1 We know what a document is.
   1.2 We can machine-read each document
   1.3 Each token is a candidate for a postings entry.
2. There is more complexity in reality

What is document?

1. What is the document unit for indexing?
   1.1 a file in a folder?
   1.2 a file containing an email thread?
   1.3 an email?
   1.4 an email with 5 attachments?
   1.5 individual sentences?
2. Answering the question "What is a document?" is not trivial
3. Precision/recall trade-off: smaller units raise precision, drop recall. why?

Parsing a document

1. Convert byte sequence into a linear sequence of characters, but
   1.1 We need to deal with format and language of each document.
   1.2 We need to determine the correct character encoding
   1.3 We need to determine format to decode the byte sequence into a character sequence MS word, zip, pdf, latex, xml (e.g., &amp). . .
   1.4 Each of these is a statistical classification problem
   1.5 Alternatively we can use heuristics
   1.6 Text is not just a linear sequence of characters (e.g., diacritics above and below letters in Arabic)
2. Some of these are a classification problem (we will study later).

Some definitions

1. Type: We call any unique word a type (the is a word type)
2. Token: An instance of a type occurring in a document (e.g., 13721 the tokens in Moby Dick).
3. Word: A delimited string of characters as it appears in the text.
4. Term : A “normalized” word (case, morphology, spelling etc); an equivalence class of words.

Tokenization

1. Text is not just a linear sequence of characters (e.g., diacritics above and below letters in Arabic)
2. What language is it in?
3. Writing system conventions?
4. Documents or their components can contain multiple languages/format; for instance a French email with a Spanish pdf attachment
5. A single index usually contains terms of several languages

Tokenization

1. Given a character sequence (and a defined document unit), we now need to determine our tokens, but, what are the correct tokens to use?

Example

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

$\boxed{\text{neill}}$

$\boxed{\text{oneill}}$

$\boxed{\text{o'neill}}$

$\boxed{\text{o'}}\ \boxed{\text{neill}}$

$\boxed{\text{o}}\ \boxed{\text{neill}}$

?

$\boxed{\text{aren't}}$

$\boxed{\text{arent}}$

$\boxed{\text{are}}\ \boxed{\text{n't}}$

$\boxed{\text{aren}}\ \boxed{\text{t}}$

?

2. The choices determine which queries will match.

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

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

Tokenization problems: Numbers

1. 3/20/91
2. 20/3/91
3. Mar 20, 1991
4. B-52
5. 100.2.86.144
6. (800) 234-2333
7. 800.234.2333
8. Older IR systems may not index numbers . . . . . . but generally it's a useful feature.

Tokenization problems: whitespace

1. No whitespace in Chinese language

   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 1 8 岁生
   日。生日派对上，莎拉波娃露出了甜美的微笑。

2. 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.

3. Compounds in Dutch, German, Swedish
   3.1 Computerlinguistik ⇒ Computer + Linguistik
   3.2 Lebensversicherungsgesellschaftsangestellter ⇒ leben + versicherung + gesellschaft + angestellter

4. Many other languages with segmentation difficulties: Finnish, Urdu, Persian, Arabic

Normalization

1. 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
2. We most commonly implicitly define equivalence classes of terms.
3. Alternatively: do asymmetric expansion
   3.1 Windows $\Rightarrow$ Windows,
   3.2 windows $\Rightarrow$ Windows, windows, window
   3.3 window $\Rightarrow$ window, windows
4. Why don't you want to put window, Window, windows, and Windows in the same equivalence class?
5. Normalization and language detection interact.
   5.1 In PETER WILL NICHT MIT, MIT $=$ mit.
   5.2 In He got his PhD from MIT, MIT $\neq$ mit.

Accents and diacritics

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

Case folding

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

Stop words

1. Stop words are 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
2. Stop word elimination used to be standard in older IR systems.
3. But you need stop words for phrase queries, e.g. “King of Denmark”
4. Most web search engines index stop words

Lemmatization

1. Reduce inflectional/variant forms to base form
2. For example
   - 2.1 Example: am, are, is ⇒ be
   - 2.2 car, cars, car's, cars' ⇒ car
   - 2.3 the boy's cars are different colors ⇒ the boy car be different color
3. Lemmatization implies doing "proper" reduction to dictionary headword form (the lemma).
4. Inflectional morphology (cutting ⇒ cut) vs. derivational morphology (destruction ⇒ destroy)

Stemming

1. Definition of stemming: Crude heuristic process that chops off the ends of words in the hope of achieving what “principled”
2. Lemmatization attempts to do with a lot of linguistic knowledge.
3. Language dependent
4. Often inflectional and derivational
   Example for derivational: automate, automatic, automation all reduce to automat
5. Most common algorithm for stemming English is Porter algorithm.
6. In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.

Exercise: What does Google do?

1. Stop words
2. Normalization
3. Tokenization
4. Lowercasing
5. Stemming
6. Non-latin alphabets
7. Umlauts
8. Compounds
9. Numbers

Exercise: Write examples for Persian language

1. Stop words
2. Normalization
3. Tokenization
4. Lowercasing
5. Stemming
6. Non-latin alphabets
7. Umlauts
8. Compounds
9. Numbers

Reuters RCV1 collection

1. Reuters RCV1 collectionis English newswire articles published in a 12-month period (1995/6)
2. It contains 800,000 documents, 400,000 terms, and 100,000,000 tokens.
3. Please see this dataset.

References

Reading

1. Chapters 1 and 2 of Information Retrieval Book[2]

[2]Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze (2008). **Introduction to Information Retrieval.** New York, NY, USA: Cambridge University Press.

References

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze (2008). **Introduction to Information Retrieval.** New York, NY, USA: Cambridge University Press.

Questions?