

Information Retrieval

Boolean information retrieval and document preprocessing

Hamid Beigy

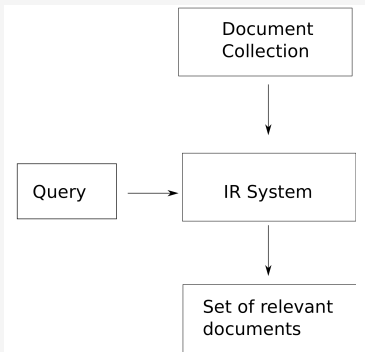
Sharif university of technology

September 30, 2019

Table of contents

Table of contents

Boolean Retrieval Model



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

Table of contents

Boolean Retrieval Model

- The 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.

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., 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

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

- 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
- $M = 500,000 \times 10^6 =$ half a trillion 0s and 1s.
- But the matrix has no more than one billion 1s.
 - Matrix is extremely sparse.
- What is a better representations?
 - We only record the 1s.

Architecture of IR systems

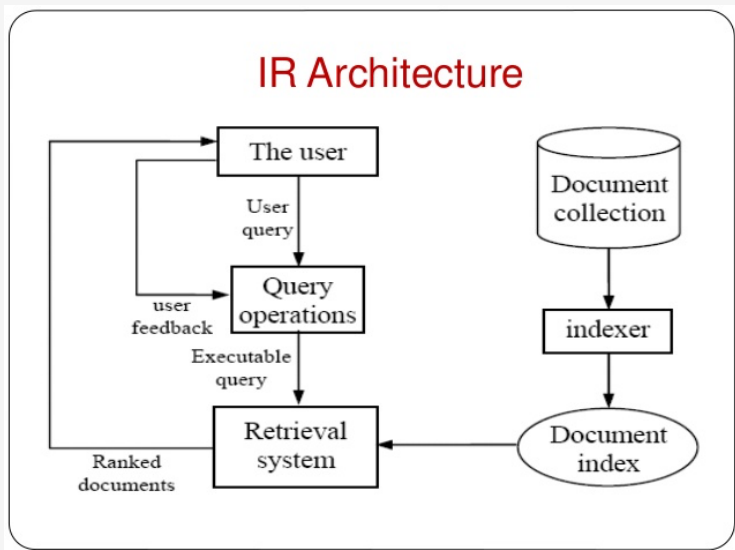
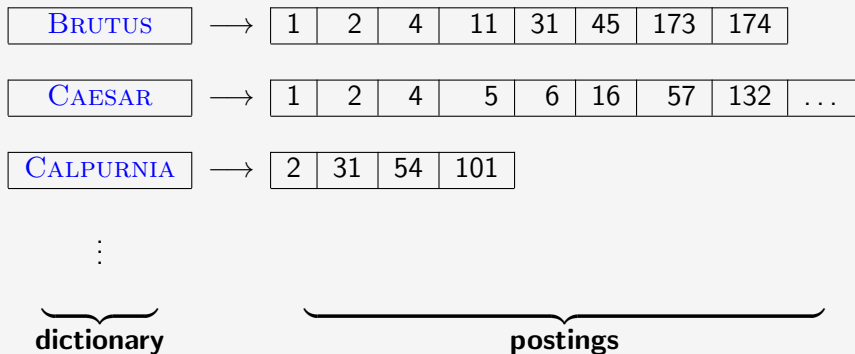


Table of contents

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.

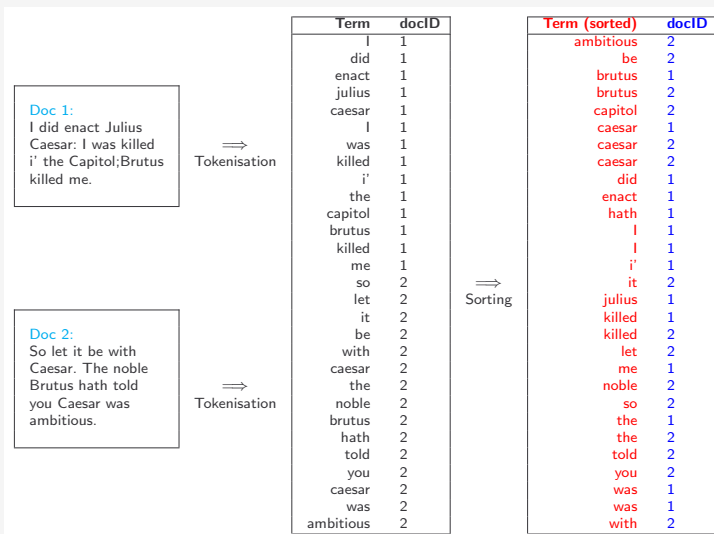
Doc 2. So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious:



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



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):
 - for more efficient Boolean searching (**we discuss later**)
 - 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

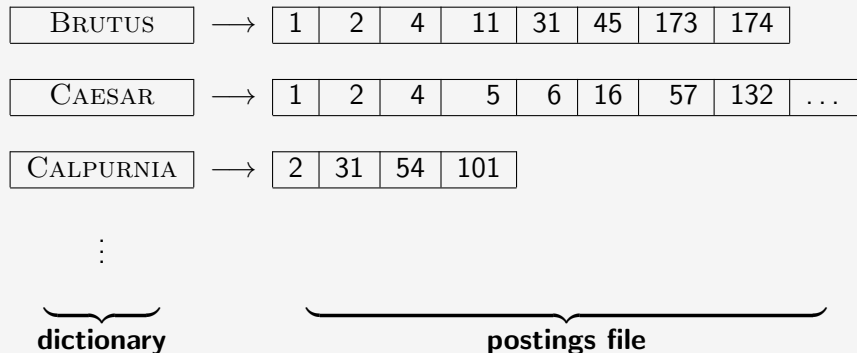


Table of contents

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 \longrightarrow $\boxed{1} \rightarrow \boxed{2} \rightarrow \boxed{4} \rightarrow \boxed{11} \rightarrow \boxed{31} \rightarrow \boxed{45} \rightarrow \boxed{173} \rightarrow \boxed{174}$

CALPURNIA \longrightarrow $\boxed{2} \rightarrow \boxed{31} \rightarrow \boxed{54} \rightarrow \boxed{101}$

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

- 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)
1  answer ← <>
2  while p1 ≠ NIL and p2 ≠ NIL
3  do if docID(p1) = docID(p2)
4     then ADD (answer, docID(p1))
5         p1 ← next(p1)
6         p2 ← next(p2)
7  if docID(p1) < docID(p2)
8     then p1 ← next(p1)
9     else p2 ← next(p2)
10 return answer

```

Brutus → 1 → 2 → 4 → 11 → 31 → 45 → 173 → 174

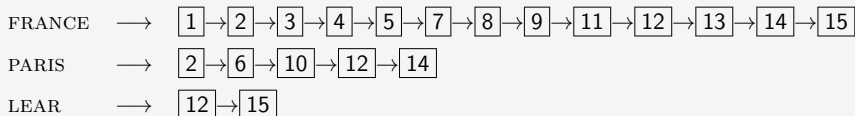
Calpurnia → 2 → 31 → 54 → 101

Intersection 2 31

Complexity of the Intersection Algorithm

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

Query processing: Exercise



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

Boolean retrieval model: Assessment

- The Boolean retrieval model can answer any query that is a Boolean expression.
 - Boolean queries are queries that 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.
- Many search systems you use are also Boolean: spotlight, email, intranet etc.

Commercially successful Boolean retrieval: Westlaw

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

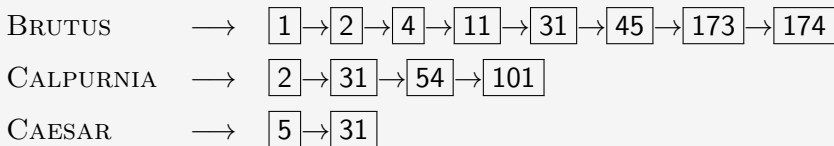
Does Google use the Boolean model?

- 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.

Table of contents

Query optimization

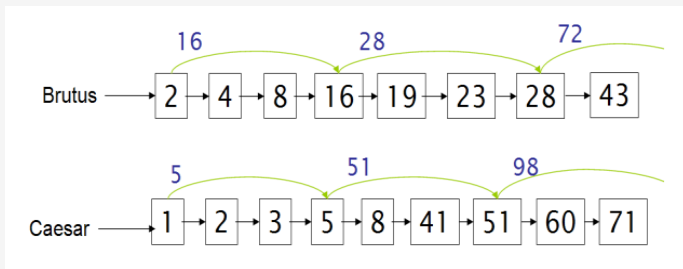
- Example query: BRUTUS AND CALPURNIA AND CAESAR
- Simple and effective optimization: **Process in order of increasing frequency**
- Start with the shortest postings list, then keep cutting further
- In this example, first CAESAR, then CALPURNIA, then BRUTUS



Optimized intersection algorithm for conjunctive queries

```
INTERSECT( $\langle t_1, \dots, t_n \rangle$ )
1  terms  $\leftarrow$  SORTBYINCREASINGFREQUENCY( $\langle t_1, \dots, t_n \rangle$ )
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

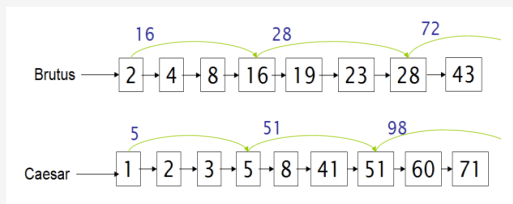
INTERSECTWITHSKIPS(p_1, p_2)

```

1  answer  $\leftarrow \langle \rangle$ 
2  while  $p_1 \neq \text{NIL}$  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?



- 1 Number of items skipped vs. frequency that skip can be taken
- 2 **More skips:** each pointer skips only a few items, but we can frequently use it, but many comparisons.
- 3 **Fewer skips:** each skip pointer skips many items, but we can not use it very often, but fewer comparisons.
- 4 This ignores the distribution of query terms.
- 5 Easy for static index; hard in dynamic environments due to updates.
- 6 How much do skip pointers help? They used to help a lot.
- 7 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:
 - biword index
 - 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
 - Searches for a single term?
 - 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

```

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  $I \leftarrow \langle \rangle$ 
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  $\text{ADD}(I, \text{pos}(pp_2))$ 
11                 else if  $\text{pos}(pp_2) > \text{pos}(pp_1)$ 
12                     then break
13                  $pp_2 \leftarrow \text{next}(pp_2)$ 
14                 while  $I \neq \langle \rangle$  and  $|I[0] - \text{pos}(pp_1)| > k$ 
15                     do  $\text{DELETE}(I[0])$ 
16                     for each  $ps \in I$ 
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

- 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

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

Table of contents

Documents

- 1 Up to now, to build an inverted index, we assumed that
 - We know what a document is.
 - We can **machine-read** each document
 - 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?
 - a file in a folder?
 - a file containing an email thread?
 - an email?
 - an email with 5 attachments?
 - individual sentences?
- 2 Answering the question "What is a document?" is not trivial
- 3 Precision/recall trade-off: smaller units raise precision, drop recall

Parsing a document

- 1 Convert byte sequence into a linear sequence of characters, but
 - We need to deal with format and language of each document.
 - We need to determine the correct character encoding
 - We need to determine format to decode the byte sequence into a character sequence
MS word, zip, pdf, latex, xml (e.g., &). . .
 - Each of these is a statistical classification problem
 - Alternatively we can use heuristics
 - 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

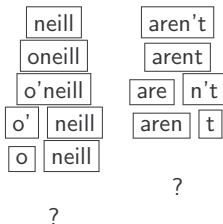
- 1 Text is not just a linear sequence of characters (e.g., diacritics above and below letters in Arabic)
- 2 What language is it in?
- 3 Writing system conventions?
- 4 Documents or their components can contain multiple languages/format; for instance a French email with a Spanish pdf attachment
- 5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

Tokenization

- 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.



- 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 its 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18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

- Computerlinguistik ⇒ Computer + Linguistik
- 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
 - Windows \Rightarrow Windows,
 - windows \Rightarrow Windows, windows, window
 - 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.
 - In **PETER WILL NICHT MIT**, MIT = mit.
 - 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
 - capitalized words in mid-sentence MIT vs. mit
 - Fed vs. fed
- 3 Its 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
 - Example: 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
- 3 Lemmatization implies doing proper reduction to dictionary headword form (the lemma).
- 4 Inflectional morphology (cutting \Rightarrow cut) vs. derivational morphology (destruction \Rightarrow 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 collection is 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.

Reading

Please read chapter 2 of Information Retrieval Book.