Text Technologies for Data Science

Indexing

Instructor: Youssef Al Hariri
Pre-lecture

• Lectures 1-4 → warmup
  • Now, it is getting more serious!

• Lab 1 → slightly different results
  • Tokenisation, stopping, stemming

• Today: two lectures on Indexing
  • expect some knowledge in binary numbers “001011101”

• Announcement of CW1 (Friday, October 6\textsuperscript{th})

• Labs → Do it On Time

• Piazza!!!!
Lecture Objectives

• Learn about and implement
• Boolean search
• Inverted index
• Positional index
Indexing Process

Documents acquisition

Document data store

Index creation

Index

web-crawling provider feeds
RSS “feeds”
desktop/email

what data do we want?

Text transformation

format conversion international?
which part contains “meaning”?
word units? stopping? stemming?

document → unique ID
what can you store?
disk space? rights?
compression?

a lookup table for quickly finding all docs containing a word

Pre-processing

Indexing

© Addison Wesley, 2008
Pre-processing output

This is an example of how the pre-processing is applied to text in information retrieval. It includes: Tokenization, Stop Word Removal, and Stemming.

- Add processed terms to the index
- What is “index”? 
Index

• How to match your term in non-linear time?
• Find/Grep:
  Sequential search for term
• Index:
  Find term locations immediately
Index

absolute error, 437
accuracy, 359
ad hoc search, 3, 280, 423
adaptive filtering, 425
adversarial information retrieval, 294
advertising, 218, 371
classifying, 371
contextual, 218–221
agglomerative clustering, 375
anchor text, 21, 56, 105, 280
API, 439, 461
architecture, 13–28
authority, 21, 111
automatic indexing, 400
background probability, see collection probability
bag of words, 345, 451
Bayes classifier, 245
Bayes Decision Rule, 245
Bayes’ Rule, 246, 343
Bayes’ rule, 342
Bayesian network, 268
bibliometrics, 120
bidding, 218
bigram, 100, 253
BigTable, 57
binary independence model, 246
blog, 111
BM25, 250–252
BM25f, 294
Boolean query, 235
Boolean query language, 24
Boolean retrieval, 235–237
boosting, 448
BPREF, 322
brute force, 331
burstiness, 254

caching, 26, 181
card catalog, 400
case folding, 87
case normalization, 87
categorization, see classification
CBIR, see content-based image retrieval
chacter encoding, 50, 119
checksum, 60
Chi-squared measure, 202
CJK (Chinese-Japanese-Korean), 50, 119
classification, 3, 339–373
faceted, 224
monothetic, 223, 374
polythetic, 223, 374
classifier, 21
crawler, 17, 32
cross-language information retrieval, 226
cross-linguistic search, see cross-language information retrieval
cross-validation, 331
Damerau-Levenshtein distance, 194
dangling link, 107
data mining, 113
database system, 459
DCG, see discounted cumulative gain
deep Web, 41, 448
delta encoding, 144
dendrogram, 375
desktop search, 3, 46
Dice’s coefficient, 192
digital reference, 447
Dirichlet smoothing, 258
discounted cumulative gain, 319
discriminative model, 284, 360
distance measure, 374
distributed hash table, 445
distributed information retrieval, 438
distribution, 23
divisive clustering, 375
document, 2
document conversion, 18
document crawler, 17
document data store, 19
document distribution, 180
document slope curve, 64
document statistics, 22
document structure, 101, 269, 459–466
document summary, 215
downcasting, 87
dumping, 366
duplicate documents, 60
dwell time, 27
dynamic page, 42
Indexing

• Search engines vs PDF find or grep?
  • Infeasible to scan large collection of text for every “search”
  • Find section that has: “UK and Scotland and Money”?! 

• Book Index
  • For each word, list of “relevant” pages 
  • Find topic in sub-linear time 

• IR Index:
  • Data structure for fast finding terms 
  • Additional optimisations could be applied
Document Vectors

• Represent documents as vectors
  • Vector → document, cell → term
  • Values: term frequency or binary (0/1)
  • All documents → collection matrix

number of occurrence of a term in a document
Inverted Index

- Represent terms as vectors
  - Vector $\rightarrow$ term, cell $\rightarrow$ document
  - Transpose of the collection matrix
  - Vector: inverted list

<table>
<thead>
<tr>
<th>Term</th>
<th>He</th>
<th>Drink</th>
<th>Ink</th>
<th>Likes</th>
<th>Pink</th>
<th>Thing</th>
<th>Wink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Value</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Value</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Value</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Value</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$\leftarrow$ D1: He likes to wink, he likes to drink
$\leftarrow$ D2: He likes to drink, and drink, and drink
$\leftarrow$ D3: The thing he likes to drink is ink
$\leftarrow$ D4: The ink he likes to drink is pink
$\leftarrow$ D5: He likes to wink, and drink pink ink
Boolean Search

• Boolean: exist / not-exist
• Multiword search: logical operators (AND, OR, NOT)
• Example
  • Collection: search Shakespeare's Collected Works
  • Boolean query: Brutus AND Caesar AND NOT Calpurnia

• Build a Term-Document Incidence Matrix
  • Which term appears in which document
  • Rows are terms
  • Columns are documents
### Collection Matrix

<table>
<thead>
<tr>
<th>Terms</th>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Brutus</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>worser</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

1 if *document* contains *term*, 0 otherwise

**Query:** Brutus AND Caesar AND NOT Calpurnia

**Apply on rows:** 110100 AND 110111 AND !(010000) = 100100
Bigger collections?

• Consider \( N = 1 \) million documents, each with about 1000 words.

• \( n = 1 \text{M} \times 1 \text{K} = 1 \text{B} \) words
  \( \Rightarrow \) Heap’s law \( \Rightarrow v \approx 500 \text{K} \)

• Matrix size = 500K unique terms \( \times 1 \text{M} \) documents
  = 0.5 trillion 0’s and 1’s entries!

• If all words appear in many documents
  \( \Rightarrow \max\{\text{count}(1’s)\} = N \times \text{doc. length} = 1 \text{B} \)

• Actually, from Zip’s law \( \Rightarrow 250k \) terms appears once!

• Collection matrix is extremely sparse. (mostly 0’s)
Inverted Index: Sparse representation

- For each term \( t \), we must store a list of all documents that contain \( t \).
  - Identify each by a **docID**, a document serial number.
Inverted Index Construction

Documents to be indexed

Token stream

Terms (modified tokens)

Inverted index

Tokenizer

Normaliser

Indexer

Friends, Romans, countrymen

Terms (modified tokens)

friend

roman

countryman

Documents to be indexed

Friends, Romans, countrymen

Terms (modified tokens)

friend

roman

countryman

Inverted index

Youssef Al Hariri, TTDS 2022/2023
**Step 1: Term Sequence**

**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
Step 2: Sorting

• Sort by:
  1) Term
  then
  2) Doc ID

<table>
<thead>
<tr>
<th>Term</th>
<th>docID</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
</tr>
<tr>
<td>enact</td>
<td>1</td>
</tr>
<tr>
<td>julius</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>i'</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>capitol</td>
<td>1</td>
</tr>
<tr>
<td>brutus</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
</tr>
<tr>
<td>so</td>
<td>2</td>
</tr>
<tr>
<td>let</td>
<td>2</td>
</tr>
<tr>
<td>it</td>
<td>2</td>
</tr>
<tr>
<td>be</td>
<td>2</td>
</tr>
<tr>
<td>with</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
</tr>
<tr>
<td>noble</td>
<td>2</td>
</tr>
<tr>
<td>brutus</td>
<td>2</td>
</tr>
<tr>
<td>hath</td>
<td>2</td>
</tr>
<tr>
<td>told</td>
<td>2</td>
</tr>
<tr>
<td>you</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
</tr>
<tr>
<td>ambitious</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>docID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ambitious</td>
<td>2</td>
</tr>
<tr>
<td>be</td>
<td>2</td>
</tr>
<tr>
<td>brutus</td>
<td>1</td>
</tr>
<tr>
<td>brutus</td>
<td>2</td>
</tr>
<tr>
<td>capitol</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>1</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>caesar</td>
<td>2</td>
</tr>
<tr>
<td>did</td>
<td>1</td>
</tr>
<tr>
<td>enact</td>
<td>1</td>
</tr>
<tr>
<td>hath</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>i'</td>
<td>1</td>
</tr>
<tr>
<td>it</td>
<td>2</td>
</tr>
<tr>
<td>julius</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>killed</td>
<td>1</td>
</tr>
<tr>
<td>let</td>
<td>2</td>
</tr>
<tr>
<td>me</td>
<td>1</td>
</tr>
<tr>
<td>noble</td>
<td>2</td>
</tr>
<tr>
<td>so</td>
<td>2</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>2</td>
</tr>
<tr>
<td>told</td>
<td>2</td>
</tr>
<tr>
<td>you</td>
<td>2</td>
</tr>
<tr>
<td>was</td>
<td>1</td>
</tr>
<tr>
<td>was</td>
<td>2</td>
</tr>
<tr>
<td>with</td>
<td>2</td>
</tr>
</tbody>
</table>
Step 3: Posting

1. Multiple term entries in a single document are merged

2. Split into Dictionary and Postings

3. Doc. Frequency (df) information is added
Inverted Index: matrix → postings

<table>
<thead>
<tr>
<th></th>
<th>he</th>
<th>drink</th>
<th>ink</th>
<th>likes</th>
<th>pink</th>
<th>thing</th>
<th>wink</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D2</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- **D1**: He likes to wink, he likes to drink
- **D2**: He likes to drink, and drink, and drink
- **D3**: The thing he likes to drink is ink
- **D4**: The ink he likes to drink is pink
- **D5**: He likes to wink, and drink pink ink

**he** → 1 2 3 4 5

**drink** → 1 2 3 4 5

**ink** → 3 4 5

**pink** → 4 5

**thing** → 3

**wink** → 1 5
**Inverted Index: with frequency**

- **Boolean**: term → DocIDs list
- **Frequency**: term → tuples (DocID,count(term)) lists

<table>
<thead>
<tr>
<th>Term</th>
<th>DocIDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>he</td>
<td>1:2</td>
</tr>
<tr>
<td></td>
<td>2:1</td>
</tr>
<tr>
<td></td>
<td>3:1</td>
</tr>
<tr>
<td></td>
<td>4:1</td>
</tr>
<tr>
<td></td>
<td>5:1</td>
</tr>
<tr>
<td>drink</td>
<td>1:1</td>
</tr>
<tr>
<td></td>
<td>2:3</td>
</tr>
<tr>
<td></td>
<td>3:1</td>
</tr>
<tr>
<td></td>
<td>4:1</td>
</tr>
<tr>
<td></td>
<td>5:1</td>
</tr>
<tr>
<td>ink</td>
<td>3:1</td>
</tr>
<tr>
<td></td>
<td>4:1</td>
</tr>
<tr>
<td></td>
<td>5:1</td>
</tr>
<tr>
<td>pink</td>
<td>4:1</td>
</tr>
<tr>
<td></td>
<td>5:1</td>
</tr>
<tr>
<td>thing</td>
<td>3:1</td>
</tr>
<tr>
<td>wink</td>
<td>1:1</td>
</tr>
<tr>
<td></td>
<td>5:1</td>
</tr>
</tbody>
</table>

Appeared in D2 3 times.
Query Processing

- Find documents matching query \{ink AND wink\}
  1. Load inverted lists for each query word
  2. Merge two postings lists \(\rightarrow\) Linear merge

- Linear merge \(\rightarrow O(n)\)
  
  \(n\): total number of posts for all query words

---

**Matches**

1: \(f(0,1)\)
3: \(f(1,0)\)
4: \(f(1,0)\)
5: \(f(1,1)\)
Phrase Search

• Find documents matching query “pink ink”
  1. Find document containing both words
  2. Both words has to be a phrase

• Bi-gram Index:

  He likes to wink, and drink pink ink

  Convert to bigrams

He_likes likes_to to_wink wink_and and_drink drink_pink pink_ink

• Bi-gram Index, issues:
  • Fast, but index size will explode!
  • What about trigram phrases?
  • What about proximity? “ink is pink”
Proximity Index

- Terms positions is embedded to the inv. Index
- Called proximity/positional index
- Enables phrase and proximity search
- Toubles (DocID, term position)

D1: He likes to wink, he likes to drink
D2: He likes to drink, and drink, and drink
D3: The thing he likes to drink is ink
D4: The ink he likes to drink is pink
D5: He likes to wink, and drink pink ink
Query Processing: Proximity

- Find documents matching query “pink ink”
  1. Use **Linear merge**
  2. Additional step: check terms positions

- **Proximity search:**
  \[ pos(\text{term}1) - pos(\text{term}2) < |w| \rightarrow #5(\text{pink,ink}) \]

**Matches**

3: \( f(1,0) = 0 \)

4: \( f(1,1) = ? = \)
  \( pos(\text{ink}) - pos(\text{pink}) == 1? \)

5: \( f(1,1) = ? = \)
  \( pos(\text{ink}) - pos(\text{pink}) == 1? \)
Proximity search: data structure

• Possible data structure:
  <term: df;
    DocNo: pos1, pos2, pos3
    DocNo: pos1, pos2, pos3
    ........ >

• Example:
  <be: 993427;
    1: 7, 18, 33, 72, 86, 231;
    2: 3, 149;
    4: 17, 191, 291, 430, 434;
    5: 363, 367, ...>
Practical
Summary

• Document Vector
• Term Vector
• Inverted Index
• Collection Matrix
• Posting
• Proximity Index
• Query Processing → Linear merge
Resources

• Textbook 1: Intro to IR, Chapter 1 & 2.4
• Textbook 2: IR in Practice, Chapter 5
• Lab 2