Text Technologies for Data Science

INFR11145

Indexing

Instructor:
Walid Magdy

07-Oct-2020

Lecture Objectives

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

- web-crawling provider feeds RSS "feeds" desktop/email
- what data do we want?
- Documents acquisition
- Text transformation
- Index creation
- Indexing
- what can you store?
- disk space? rights?
- compression?
- a lookup table for quickly finding all docs containing a word

**Pre-processing output**

This is an example sentence of how the pre-processing is applied to text in information retrieval. It includes: tokenization, stop word removal, and stemming.

- Add processed terms to 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

Book Index

Index

absolute error, 437
accuracy, 359
all-text search, 3, 280, 423
adversarial information retrieval, 294
advertising, 210, 371
classifying, 350
contextual, 218–221
interactive clustering, 375
anchor text, 21, 56, 105, 298
API, 439, 561
architecture, 15–28
authority, 21, 111
automatic indexing, 400
background probability, 271
bag of words, 345, 451
Bayes classifier, 245
Bayes Decision Rule, 295
Bayes’ Rule, 246, 290
Bayes’ rule, 942
Bayesian network, 348
bibliometrics, 120
building, 218
bigram, 181, 219
BigData, 57
binary independence model, 246
blog, 111
BM25, 230–252
BM25+, 294
Boolean query, 235
Boolean query language, 24
Boolean retrieval, 233–237
boosting, 448
BPR, 322
Breslow fare, 331
iteration, 254
catching, 26, 181
cart-catalog, 400
case folding, 87
case-normalization, 87
correlation, see classification
CIRL, see content-based image retrieval
coreference, 50, 119
cronbach, 60
Chi-squared measure, 202
CISR (Chinese-Japanese-Korean), 50, 119
classification, 3, 230–237
cooccurrence, 224
monotonic, 223, 374
polynomial, 223, 374
classifier, 21

532 Index
clickthrough, 6, 27, 207, 285, 306
classifier, see email-language information retrieval
classifier hypothesis, 309
classifier-based retrieval, 301
clustering, 32, 222–223, 339, 373
covariance, 74, 191
code page, 50
collaborative filtering, 412
collaborative search, 420
collaboration, 3
collection language model, 256
collection probability, 256, 344
collaboration, 74
code frequency, 437
combining evidence, 267–283
combining vectors, 448
CombM2, 441
community-based question answering, 413
complex field clusters, 379
compression, 96
content, 141
core, 162
contingency random field, 122
correlation, see content
content-based image retrieval, 475
context, 115, 201, 211–214
context vector, 206, 405
contingency table, 248
controlling vocabulary, 109, 401
correlation, 49
correlation level match, 257
county, 6
corpus, 239
coverage, 8
CQA, 415
crawls, 17, 32
cross-document information retrieval, 226
cross-lingual search, see cross-language information retrieval
cross-validation, 331
Demographic information, 194
dangling links, 187
determining, 115
database entries, 459
DCG, see discounted cumulative gain
depth Web, 4, 448
definition, 144
dependencies, 375
desktop search, 3, 66
dev's coefficient, 332
digital reference, 447
Dijkstra's algorithm, 258
discounted cumulative gain, 339
discriminative model, 184, 260
disease, 374
discussion, 445
discussion, 27
discussion, 354
discussion, 375
document, 2
document creation, 88
document content, 47
document data store, 19
document distribution, 180
document drop curve, 64
document statistics, 22
domains scores, 110, 269, 495–466
downweight, 87
downweight, 656
duplicate documents, 69
dwell time, 27
dynamic page, 62
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

<table>
<thead>
<tr>
<th>he</th>
<th>drink</th>
<th>ink</th>
<th>likes</th>
<th>pink</th>
<th>think</th>
<th>wink</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</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

number of occurrence of a term in a document
**Inverted Index**

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

<table>
<thead>
<tr>
<th></th>
<th>drink</th>
<th>ink</th>
<th>likes</th>
<th>pink</th>
<th>think</th>
<th>wink</th>
</tr>
</thead>
</table>
| D1: He likes to wink, he likes to drink | 2    | 1    | 0     | 2    | 0     | 0    | 1    | D1: He likes to wink, he likes to drink
| D2: He likes to drink, and drink, and drink | 1    | 3    | 0     | 1    | 0     | 0    | 0    | D2: He likes to drink, and drink, and drink
| D3: The thing he likes to drink is ink | 1    | 1    | 1     | 1    | 0     | 1    | 0    | D3: The thing he likes to drink is ink
| D4: The ink he likes to drink is pink | 1    | 1    | 1     | 1    | 1     | 0    | 0    | D4: The ink he likes to drink is pink
| D5: He likes to wink, and drink pink ink | 1    | 1    | 1     | 1    | 1     | 0    | 1    | 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>

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.
- $n = 1M \times 1K = 1B$ words
  - Heap’s law $\rightarrow v \approx 500K$
- Matrix size = 500K unique terms x 1M documents = 0.5 trillion 0’s and 1’s entries!
- If all words appear in many documents $\rightarrow$ max{count(1’s)} = $N \times$ doc. length = 1B
- 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.

**Dictionary**

<table>
<thead>
<tr>
<th></th>
<th>Postings List</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brutus</td>
<td>1 2 4 11 31 45 173</td>
</tr>
<tr>
<td>Caesar</td>
<td>1 2 4 5 6 16 57 132</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>2 31 54 101</td>
</tr>
</tbody>
</table>

**Posting**

**Inverted Index Construction**

1. **Documents to be indexed**: Friends, Romans, countrymen
2. **Tokenizer**:
   - **Token stream**: Friends Romans Countrymen
3. **Normaliser**:
   - **Terms (modified tokens)**: friend roman countryman
4. **Indexer**:
   - **Inverted index**:
     - friend: 2 4
     - roman: 1 2
     - countryman: 3 9
### 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
  2) Doc ID

---

**Documents with their corresponding terms and doc IDs**

<table>
<thead>
<tr>
<th>Term</th>
<th>Doc ID</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>Doc ID</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>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

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
**Inverted Index: with frequency**

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

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

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

<table>
<thead>
<tr>
<th>Term</th>
<th>3:1</th>
<th>4:1</th>
<th>5:1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ink</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>wink</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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)

- **he**
  1:2, 2:1, 3:1, 4:1, 5:1

- **drink**
  1:1, 2:3, 3:1, 4:1, 5:1

- **he**
  1:1, 1:5, 2:1, 3:3, 4:3, 5:1

- **drink**
  1:8, 2:4, 2:6, 2:8, 3:6, 4:5, 5:6

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:
  \[ \text{pos}(\text{term1}) - \text{pos}(\text{term2}) < |w| \Rightarrow \#5(\text{pink}, \text{ink}) \]

\[ \begin{array}{c}
\text{ink} \\
3, 8 \\
4, 2 \\
5, 8 \\
\end{array} \quad \begin{array}{c}
\text{pink} \\
4, 8 \\
5, 7 \\
\end{array} \]

Matches

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

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

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

Proximity search: data structure

- Possible data structure:
  
  \[
  <\text{term}: \text{df}; \quad \text{DocNo}: \text{pos1}, \text{pos2}, \text{pos3}; \quad \text{DocNo}: \text{pos1}, \text{pos2}, \text{pos3}; \\
  \ldots... >
  \]

- Example:
  
  \[
  <\text{be}: 993427; \quad 1: 7, 18, 33, 72, 86, 231; \quad 2: 3, 149; \quad 4: 17, 191, 291, 430, 434; \quad 5: 363, 367, ... >
  \]
**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