Text Technologies for Data Science
INFR11145

Indexing

Instructor:
Walid Magdy

Lecture Objectives

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

- Documents acquisition
- Document data store
- Index creation
- Index

Pre-processing output

- Example sentence
- Pre process
- Applied text information retrieval
- Include 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
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></th>
<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>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>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>D4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</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>

\( \text{number of occurrence of a term in a document} \)
Inverted Index

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

<table>
<thead>
<tr>
<th>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>0</td>
<td>1</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>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</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

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.
- $n = 1M \times 1K = 1B$ 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 \texttt{docID}, a document serial number

<table>
<thead>
<tr>
<th>Term</th>
<th>Doc number</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, 37, 132</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>2, 31, 54, 101</td>
</tr>
</tbody>
</table>

Inverted Index Construction

Documents to be indexed

Token stream

Terms (modified tokens)

Inverted index

Friends, Romans, countrymen

friend, roman, countryman

friend, roman, countryman

friend, roman, countryman
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

Sorting

<table>
<thead>
<tr>
<th>Term</th>
<th>docID</th>
</tr>
</thead>
<tbody>
<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>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>be</td>
<td>2</td>
</tr>
<tr>
<td>brutus</td>
<td>1</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>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>you</td>
<td>2</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 \(\rightarrow\) postings

\[
\begin{array}{cccccccc}
\text{he} & \text{drink} & \text{ink} & \text{likes} & \text{pink} & \text{think} & \text{wink} \\
2 & 1 & 0 & 2 & 0 & 0 & 1 \\
1 & 3 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 \\
\end{array}
\]

\(\xrightarrow{\text{D1}}\): He likes to wink, he likes to drink

\(\xrightarrow{\text{D2}}\): He likes to drink, and drink, and drink

\(\xrightarrow{\text{D3}}\): The thing he likes to drink is ink

\(\xrightarrow{\text{D4}}\): The ink he likes to drink is pink

\(\xrightarrow{\text{D5}}\): He likes to wink, and drink pink ink
**Inverted Index: with frequency**

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

```
he  1:2  2:1  3:1  4:1  5:1
drink  1:1  2:3  3:1  4:1  5:1
ink  3:1  4:1  5:1
pink  4:1  5:1
thing  3:1
wink  1:1  5:1
```

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)
  
  \[ n: \text{total number of posts for all query words} \]

```
ink  3:1  4:1  5:1
wink  1:1  5:1
```

**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  
  \[
  \text{He\_likes\_to\_to\_wink\_wink\_and\_and\_drink\_drink\_pink\_pink\_ink}
  \]

- 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)

\[
\begin{align*}
\text{he} & \rightarrow 1:2 \quad 2:1 \quad 3:1 \quad 4:1 \quad 5:1 \\
\text{drink} & \rightarrow 1:1 \quad 2:3 \quad 3:1 \quad 4:1 \quad 5:1 \\
\end{align*}
\]

\[
\begin{align*}
\text{D1: He likes to wink, he likes to drink} \\
\text{D2: He likes to drink, and drink, and drink} \\
\text{D3: The thing he likes to drink is ink} \\
\text{D4: The ink he likes to drink is pink} \\
\text{D5: He likes to wink, and drink pink ink}
\end{align*}
\]
Query Processing: Proximity

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

- Proximity search:
  \( \text{pos(term1)} - \text{pos(term2)} < |w| \rightarrow \#(\text{pink,ink}) \)

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

\[
\begin{array}{c}
pink \\
\Uparrow
\end{array}
\begin{array}{c}
4, 8 \\
5, 7
\end{array}
\]

Matches

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

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

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

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, ...>

Walid Magdy, TTDS 2022/2023

THE UNIVERSITY OF EDINBURGH
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