Introduction to Information Retrieval: IR Basics and Evaluation

Prof. Srijan Kumar
Logistics

- **Class size:** Due to huge demand, class size has been increased to 85
- **Piazza:** Please join
  - [https://piazza.com/class/spring2020/cse6240/](https://piazza.com/class/spring2020/cse6240/) (same link as before)
- **Canvas:** Logistical issues being resolved now
- **Project:**
  - Example datasets and sample projects will be released by Thursday evening
  - Teams due by Jan 20
Today’s Class

• Web is a collection of documents
  – E.g., web pages, social media posts

• Web is a network
  – E.g., the hyperlink network of websites, network of people on social networks

• Web is a set of applications
  – E.g., e-commerce platforms, content sharing, streaming services

Some slides from today’s lecture are inspired from Prof. Hongyuan Zha’s past offerings of this course
Today’s Class: Part 1

• Web is a collection of documents
  1. Process documents for search and retrieval
  2. Quantifying the quality of retrieval
Search and Retrieval are Everywhere

- **Web search engines: Querying for documents on the web**
  - Google, Bing, Yahoo Search
- **E-commerce platforms: Querying for products on the platform**
  - Amazon, eBay
- **In-house enterprise: Querying for documents internal to the enterprise**
  - Universities, Companies
• **Goal:** Index documents to be easily searchable

• **Steps to index documents:**
  1. **Collect** the documents to be indexed
  2. **Tokenize** the text
  3. **Normalize** of the text (linguistic processing)
  4. **Index** the text: Inverted Indexing
Processing Document Collections

Documents to be indexed.

Token stream.

Modified tokens.

Inverted index.

Tokenizer

Linguistic modules

Indexer

Tokenization and linguistic processing determine the terms considered for retrieval

[Diagram]

- **Documents to be indexed:**
  - Input documents for indexing.

- **Token stream:**
  - Output of the tokenizer. It includes tokens like "Friends", "Romans", and "Countrymen".

- **Modified tokens:**
  - Tokens are processed and modified by linguistic modules. Examples include "friend", "roman", and "countryman".

- **Inverted index:**
  - Output of the indexer. It shows term-document frequencies, such as "friend" with frequencies 2 and 4, "roman" with 1 and 2, and "countryman" with 13 and 16.
Processing Document Collections

Tokenization and linguistic processing determine the terms considered for retrieval.
Tokenization

• Tokenization formats the text by chopping it up into pieces, called **tokens**
  – E.g., remove punctuations and split on white spaces
  – Georgia-Tech → Georgia Tech

• However, **tokenization can give unwanted results**
  – San Francisco → “San” “Francisco”
  – Hewlett-Packard → Hewlett Packard
  – **Dates:** 01/08/2020 → 01 08 2020
  – **Phone number:** (800) 111-1111 → 800 111 1111
  – **Emails:** srijan@cs.stanford.edu → srijan cs stanford edu

• Such splits can result in poor retrieval results
Tokenization: What To Do?

• So, what should one do?

• Come up with **regular expression rules**
  – E.g., only split if the next word starts with a lowercase letter

• **Has to be language specific:** English rules not applicable to all other languages
  – E.g., French: *L’ensemble*
  – German: *Computerlinguistik* means ‘computational linguistics’
Processing Document Collections

Tokenization and linguistic processing determine the terms considered for retrieval.
Text Normalization: Why is it Needed?

- The same text can be written in many ways
  - USA vs U.S.A. vs usa vs Usa
- We need some way to create a **unified representation** to match them
- The **same normalization** is required for the query and the documents
Text Normalization: Other Languages

- **Accents**: resume vs résumé
- **Most important criteria**: How are your users likely to write their queries?
- Even in languages where the accents are the norm, users often not type them, or the input device is not convenient
- **German**: Tuebingen vs. Tübingen
  – should be the same
- **Dates**: July 30 vs. 7/30
Text Normalization Step 1: Case Folding

• Reduce all letters to lower case
  – exception: upper case (in mid-sentence?)

• Often best to lower case everything, since users tend to use lowercase regardless of the correct capitalization

• However, many proper nouns are derived from common nouns
  – General Motors, Associated Press

• We can create advanced solutions (later): bigrams, n-grams
Text Normalization Step 2: Remove Stop Words

• With a stop-word list, one excludes from the dictionary the most common words
  – They have little semantic content: the, a, and, to
  – They take a lot of space: 30% of postings for top 30

• Fewer stop words:
  – Can use good compression techniques
  – Good query optimization techniques mean one pays little at query time for including stop words
Text Normalization Step 2: Remove Stop Words

• However, stop words can be needed for:
  – Phrase queries: "King of Prussia"
  – (Song) titles etc.: "Let it be", "To be or not to be"
  – Relational queries: "flights to London"
Text Normalization Step 3: Stemming

• **Key idea:** Derive the base form of words, i.e. root form, to standardize their use
  – Reduce terms to their “roots” before indexing

• **Variations of words do not add value for retrieval**
  – *Grammatical* variations: organize, organizes, organizing
  – *Derivational* variations: democracy, democratic, democratization

• “**Stemming**” suggest crude suffix chopping
  – Again, language dependent
  – E.g., organize, organizes, organizing → organiz
Text Normalization Step 3: Stemming

for example
compressed and compression are both accepted as equivalent to compress

for example compress and compress are both accept as equivalent to compress
Porter’s Stemmer

• Most commonly used stemmer for English
  – Empirical evidence: as good as other stemmers

• Conventions + five phases of reductions
  – phases applied sequentially
  – each phase consists of a set of commands
  – sample convention: of the rules in a compound command, select the one that applies to the longest suffix
## Porter’s Stemmer: Rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSES → SS</td>
<td>caresses → caress</td>
</tr>
<tr>
<td>IES → I</td>
<td>ponies → poni</td>
</tr>
<tr>
<td>SS → SS</td>
<td>caress → caress</td>
</tr>
<tr>
<td>S →</td>
<td>cats → cat</td>
</tr>
</tbody>
</table>
Processing Document Collections

Tokenization and linguistic processing determine the terms considered for retrieval.
Scoring and Ranking Documents

• Ranked list of documents:
  – Order the documents most likely to be relevant to the searcher
  – It does not matter how large the retrieved set is

• How can we rank-order the docs in the collection with respect to a query?

• Begin with a perfect world – no spammers
  – Nobody stuffing keywords into a doc to make it match queries
Techniques For Indexing

1. Term-Document Incidence Matrix
2. Inverted Index
3. Positional Index
4. TF-IDF
Technique 1: Term-Document Incidence Matrix

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

- For Boolean query “Brutus AND Caesar AND NOT Calpurnia”
  - 110100 AND 110111 AND 101111 = 100100

- **Not scalable**: Billions of terms and millions of documents
Technique 2: Inverted Index

- An inverted index consists of a dictionary and postings.
- For each term T in the dictionary, we store a list of documents containing T.

<table>
<thead>
<tr>
<th>Dictionary</th>
<th>Postings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brutus</td>
<td>1 2 4 11 31 45 173 174</td>
</tr>
<tr>
<td>Caesar</td>
<td>1 2 4 5 6 16 57 ...</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>2 31 54 101</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
# Building an Inverted Index

1. **Tokenize documents**
   - I
   - did
   - enact
   - julius
   - casear
   - I
   - was
   - killed
   - i’
   - the
   - capitol
   - brutus
   - killed
   - me
   - so
   - let
   - it
   - be
   - with
   - caesar
   - the
   - noble
   - brutus
   - hath
   - told
   - you
   - caesar
   - was
   - ambitious

2. **Sort alphabetically**
   - ambitious
   - be
   - brutus
   - caesar
   - caesar
   - caesar
   - caesar
   - did
   - enact
   - hath
   - I
   - i’
   - it
   - julius
   - killed
   - killed
   - me
   - me
   - noble
   - so
   - the
   - the
   - told
   - you
   - was
   - was
   - was
   - with

3. **Compress using counts/term frequency**
   - ambitious 2
   - be 2
   - brutus 2
   - caesar 2
   - caesar 2
   - caesar 2
   - caesar 2
   - did 1
   - enact 1
   - hath 1
   - I 1
   - i’ 1
   - it 1
   - julius 1
   - killed 2
   - killed 1
   - me 2
   - me 1
   - noble 2
   - so 1
   - the 3
   - the 2
   - told 2
   - you 2
   - was 1
   - was 1
   - was 2
   - with 2

Srijan Kumar, Georgia Tech, CSE6240 Spring 2020: Web Search and Text Mining
Building an Inverted Index II

Compress by creating a list of documents that have the term
Retrieval with Inverted Index

- **Example query:** Brutus AND Calpurnia
- **Steps:**
  - Locate Brutus in the Dictionary
  - Retrieve its postings
  - Locate Calpurnia in the Dictionary
  - Retrieve its postings
  - Intersect the two postings lists
Algorithm to Intersect/Merge Lists

• Postings in sorted order, complexity $O(x + y)$

```
INTERSECT(p, q)
1    answer ← ∅
2    while $p \neq$ NIL and $q \neq$ NIL
3        do if docID[p] = docID[q]
4            then ADD(answer, docID[p])
5            p ← next[p]
6            q ← next[q]
7        else if docID[p] < docID[q]
8            then p ← next[p]
9            else q ← next[q]
10    return answer
```