Agenda for today

- Information Retrieval Basics
  - Term-document matrix
  - Inverted indices
  - Boolean retrieval, index intersection
- Additional topics on terms and postings
  - Faster intersection of posting lists
  - Positional indices
  - Tokenization and normalization

What is IR

Definition from Manning, Raghavan and Schütze:

Information retrieval (IR) is:

finding **material** (usually documents)
of an **unstructured** nature (usually text)
that satisfies an **information need**
from within **large collections** (usually stored on computers).

Traditionally...

Material

- Most retrieved data currently documents consisting largely of text
  - Disparate file formats (MS Word, PDF, etc.)
  - Various text encodings (e.g., ASCII, UTF-8, etc.)
  - Inconsistently formatted
  - Multilingual
- Multi-modal retrieval increasingly important
  - Images, speech, video, smells, etc.
  - Each modality introduces its own complications
  - Mixtures of modalities present opportunities and challenges

Unstructured

- Definition from Manning, Raghavan and Schütze:

  " data which does not have clear, semantically overt, easy-for-a-computer structure "

- Does not mean there is no structure in the data
  - Document structure (headings, paragraphs, lists, etc.)
  - Explicit document markup formatting
  - Linguistic structure (hidden)
- Rather, not structured rigidly like a relational database

Information need / large collection

- Fundamental model
  - Collection/corpus of "documents"
  - Each document consists of some number of "terms"
- User has an information need, presented as a query
  - Map query to "terms"; search for "documents" containing "terms"
- Many possible realizations of this model:
  - What's a "document" (unit of retrieval)?
  - What's a "term"? (typically a word/token)
- Whatever the definition: find documents containing terms

Exact match problem

- Nearly the same problem as exact match in bioinformatics
  - Given a relatively short *pattern* (term)
  - Find instances of the pattern in a large *text*
- Here we have documents, sort of like bins in the text
  - Do not necessarily need exact position, just document ID
- Algorithms can either focus on the *pattern* or the *text*
  - Preprocess the pattern for sub-linear scan of whole text
  - Preprocess the text if it is fixed (e.g., suffix tree)
- In IR, document sets large enough that *grep* not an option

Term-document matrix

- Starting point for thinking about indexing
- Establish list of terms (lexicon, dictionary) in the corpus
- Create a matrix with rows=terms and columns=documents
- Cell (t, d) has 1 if term t occurs in d; 0 otherwise:

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Antony	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	
...							

Boolean Retrieval

- Query: Brutus AND Caesar AND NOT Calpurnia
- Determine the bit vector (row) for each term in query:

Brutus:	`110100`
Caesar:	`110111`
Calpurnia:	`010000`

- Take the complement of Calpurnia: `101111`
- Perform a bitwise AND:

Brutus	`110100`
Caesar:	`110111`
NOT Calpurnia:	`101111`
Answer	`100100`

Term-document matrix

- Limited in key ways
  - Matrix grows too large for standard sized problems

    e.g., hundreds of thousands of terms, millions of documents
  - Fortunately very sparse, amenable to sparse representations
  - Great for simple query operators, others not feasible

    e.g., two query words occurring “near” each other
  - No means to rank the resulting matching document set
- Will move towards richer representations, beginning with “inverted” index

Inverted index

- Sparse representation of the term-document matrix
- Sorted list of document IDs ("postings") for each term

Dictionary		Postings
Brutus	$\longrightarrow$	1 | 2 | 4 | 11 | 31 | 45 | 173 | 174
Caesar	$\longrightarrow$	1 | 2 | 4 | 5 | 6 | 16 | 57 | 132 | ...
Calpurnia	$\longrightarrow$	2 | 31 | 54 | 101
$\vdots$		

- Determine results by intersecting postings

Building inverted indices

- Four basic steps to build an inverted index for a collection
  - Collect the documents to be collected (unit of retrieval)
  - Tokenize the collection; documents become lists of tokens
  - Text preprocessing (decasing, normalization, stemming, etc.)
  - Index the tokenized, normalized collection
    - Indices typically include document frequency (# of postings)
- Rather than simple bitwise intersection, now intersecting list
  - Assume sorted lists; simple, efficient algorithms
  - Various methods to improve typical complexity

Merge algorithm

- Merge algorithm: scan through sorted lists simultaneously; advance pointer on the lesser document ID

Caesar	$\longrightarrow$	1	2	4	5	6	16	57	132	...
Calpurnia	$\longrightarrow$	2	31	54	101					

Merge algorithm

- Merge algorithm: scan through sorted lists simultaneously; advance pointer on the lesser document ID

Caesar	$\longrightarrow$	1	2	4	5	6	16	57	132	...
Calpurnia	$\longrightarrow$	2	31	54	101					

```
INTERSECT(p1, p2)
answer ← ⟨ ⟩
while p1 ≠ NIL and p2 ≠ NIL
do if docID(p1) = docID(p2)
      then ADD(answer, docID(p1))
           p1 ← next(p1)
           p2 ← next(p2)
      else if docID(p1) < docID(p2)
             then p1 ← next(p1)
             else p2 ← next(p2)
return answer
```

Merge algorithm

- For lists of length m and n, complexity $O(m + n)$
  - Bounded by $O(N)$ for a collection of N documents
- Non-optimal in certain scenarios
  - e.g., very small list with a large list
- Various optimizations can be carried out to improve typical case
  - Fast binary search methods for long list with very short list
    - Complexity $O(m \log n)$ instead of $O(m + n)$
  - Optimizations on complex queries to determine most restrictive
    - Most restrictive first: (Brutus AND Calpurnia) AND Caesar

Data structures for index

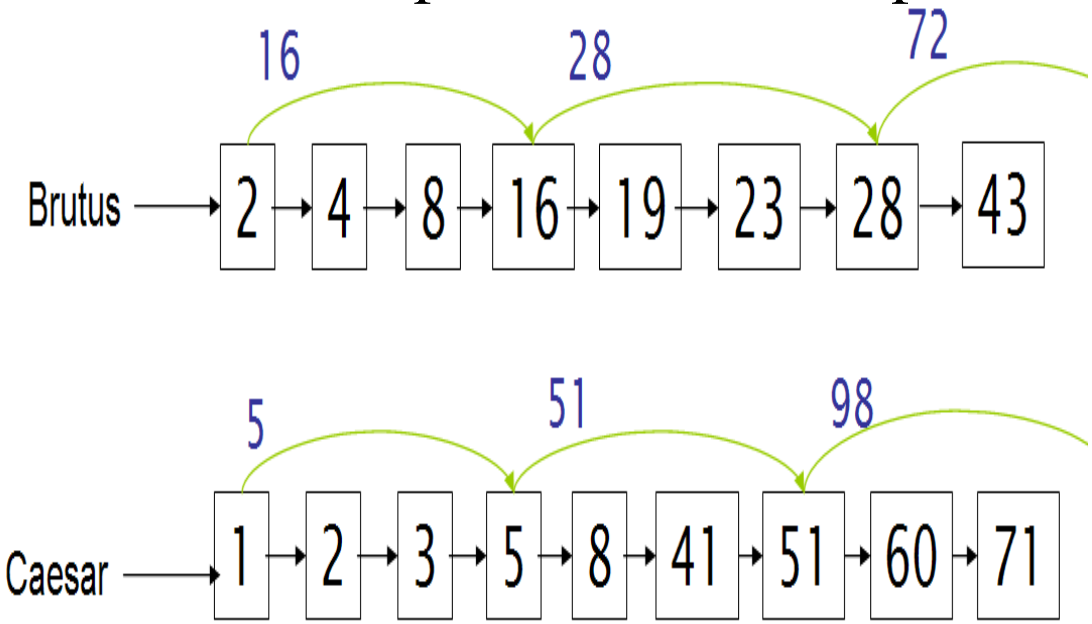
- Typically store document frequency (DF) with postings list:

term	**doc. freq.**	$\rightarrow$	**postings lists**
ambitious	1	$\rightarrow$	$\boxed{2}$
be	1	$\rightarrow$	$\boxed{2}$
brutus	2	$\rightarrow$	$\boxed{1} \rightarrow \boxed{2}$
capitol	1	$\rightarrow$	$\boxed{1}$
caesar	2	$\rightarrow$	$\boxed{1} \rightarrow \boxed{2}$

- Can store postings lists in a couple of ways
  - Linked list: easy insertion of new doc IDs
  - Variable length arrays: space efficient, cache friendly
- DF provides size of array of posting list; good for binary search
  - Hence structure of choice for query optimization

Skip lists

- One linked list optimization in chapter 2 of Manning et al.:



- Check skip link if present to skip multiple entries
- Heuristic for skip placement in list of length L:
  - Evenly space $\sqrt{L}$ skip links

Pros/cons of Boolean model

- Boolean retrieval contrasts with “Ranked retrieval” approaches
  - Structured versus unstructured queries
  - Recall versus precision: Boolean can easily over-specify query, resulting in no returned results (rare in current search engines)
  - Returning nothing not as useful for query reformulation
- Expert users often prefer Boolean systems to ranked retrieval
  - User feels like they have more control of system behavior
  - Validation of the expertise of the user: librarian, analysts, etc.
- Represents a big divide between “classical” and “modern” IR

Classical vs. Modern IR

- Back in the day
  - Highly trained professional system users (librarians, analysts)
  - Computers and data transmission were slow and expensive
    - e.g., many systems charged a per-query fee
  - Users typically had well-defined information needs
- Today
  - Typical users are not professionals, i.e., untrained
  - Computers are fast and basically free
  - Information needs are often much less well-defined
    - “what TV should I buy”
    - “cases on employment law involving trade secret disclosures in the semi-conductor industry in which the plaintiff blah blah blah”

Extended Boolean model

- Boolean model just supports AND, OR, NOT and presence/absence
- Ultimately want free-text queries of the sort we know and love
  - More complicated queries: collocations, “proximity”
- Also some kind of ranking model to sort resulting documents
- Extended Boolean model takes term weighting into account
  - How often does the term occur in the particular document?
  - Is that frequency surprising given distribution of term?
- Requires storing more information in index
  - Which can also permit positional queries (“within three words”)

Positional indices

- Augment inverted indices with positions of each token:

  word, DF: $\langle$ docID, TF: $\langle$pos$_1$, pos$_2$, ... , pos$_k\rangle$; ... $\rangle$

  to, 993427:
  $\langle$ 1, 6: $\langle 7, 18, 33, 72, 86, 231\rangle$;
  2, 5: $\langle 1, 17, 74, 222, 255\rangle$;
  4, 5: $\langle 8, 16, 190, 429, 433\rangle$;
  5, 2: $\langle 363, 367\rangle$;
  7, 3: $\langle 13, 23, 191\rangle; \ldots \rangle$

  be, 178239:
  $\langle$ 1, 2: $\langle 17, 25\rangle$;
  4, 5: $\langle 17, 191, 291, 430, 434\rangle$;
  5, 3: $\langle 14, 19, 101\rangle; \ldots \rangle$

- Much larger (but optimizations to come); intersection also more expensive

What do positional indices buy us?

- Proximity-based queries
  - employment \3 place (within three words on either side)
  - “New York”; “Hong Kong”; “Steven Bedrick”
    (without having to include these n-grams in the index itself)
  - Algorithms in Manning et al. text (offset arithmetic req’d)
- Have term frequency available, for term weighting
  - Obviates the need for explicit stop-word list creation
  - Provides the means for assessing query relevance of match

Ranking retrieval results

- Huge topic, one we'll return to a lot over the course
- Key intuition:

  a document that frequently mentions a query term should probably be ranked higher than a document that only mentions it once
- However, not every query term is equally important
  - Closed-class words ('the', 'of', ...) in most English documents
  - Corpus-specific frequent terms have similar problems
    - e.g., "auto" in a corpus related to cars; "brain" or "cell" in neuroscience articles
- Must also consider the document frequency of a term

TF*IDF

- TF=term frequency; IDF=inverse document frequency
- A start towards weighting terms: favor high TF, low DF
  - Use DF rather than total count, less impacted by outliers
- In practice, IDF of term calculated as $\log \frac{\mathrm{N}}{\mathrm{DF}}$ for N documents
- Score of frequently occurring words severely penalized
- IDF can be seen as a kind of "expected document frequency"
- Favor terms occurring significantly more frequently that typical
  - Other metrics (log likelihood or odds ratios) more directly measure this

Ranking documents

- Given a query $\boldsymbol{q}$ and a document $\boldsymbol{d}$, score the document by summing tf-idf for every term $\boldsymbol{t}$ in $\boldsymbol{q}$, i.e., in Manning et al.:

$$\text{Score}(\boldsymbol{q}, \boldsymbol{d}) = \sum_{\boldsymbol{t} \in \boldsymbol{q}} \text{tf-idf}_{\boldsymbol{t},\boldsymbol{d}} \tag{6.9}$$

- This formulation allows for alternatives to tf-idf
  - e.g., vector space models, to be covered next lecture
- Methods still depend on key initial questions:
  - What is a document?
  - What is a term?

Terms and documents

- Chapter 2 of Manning et al. covers important issues in defining "documents" (unit of retrieval) and "terms" in vocabulary
- Many issues that we deal with in other courses
  - Document format normalization (mapping to text sequences)
  - Language identification
  - Tokenization and segmentation
  - Term normalization (e.g., de-casing, stemming, lemmatization)
- Mapping to an internal representation:

  Consistent normalization more important than human readability

Many non-English phenomena

- Ordering of characters within script, e.g., Arabic diacritics

  كِتَابٌ ⇐ ٌ ب ا ت ِ ك
  un b ā t i k (complicated by, e.g., numbers)
  /kitābun/ *'a book'*

- Tokenization of nominal compounds in German:

  Lebensversicherungsgesellschaftsangestellter

  "life insurance company employee"

- Word segmentation in East Asian Languages, e.g., Chinese:

  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。

Normalization

- Mapping of multiple terms to a single class
- Case-folding is an obvious case: both *This* and *this* are the same
- Prefer easy, deterministic transforms, e.g., remove hyphens or punctuation: U.S.A. $\rightarrow$ USA; Case-folding $\rightarrow$ Casefolding
- Map symbols with diacritics to the base symbol without
- Date and number normalization
- Stemming and lemmatization (rules for mapping term classes)
  - e.g., Porter stemmer suffix rule in English: ies $\rightarrow$ i
    thus ponies $\rightarrow$ poni (correct root 'pony' also maps to 'poni')
- Doesn't matter that the mapping is wrong (internal representation)

What have we covered

- Figure out your terms and your ‘document’ granularity
  - Variously complex pre-processing required
- Simple Boolean term-document matrix
- Sparse matrix representation $\rightarrow$ inverted index
- Merge algorithm for intersection postings list
- Optimizing queries and structures: binary search and skip lists
- Extended Boolean model: keep DF, TF and offsets
  - positional indices
- Calculate tf-idf (or other scores) and rank retrieval results

Next week

- IR Models
  - Vector space models
  - Language models
- Index construction/optimization/compression