Mid-term exam

March 23rd

In-class exam

Closed book closed notes!

Text Retrieval and Mining

Query

- Which plays of Shakespeare contain the words ***Brutus*** *AND* ***Caesar*** but *NOT* ***Calpurnia***?
- One could grep all of Shakespeare's plays for ***Brutus*** and ***Caesar,*** then strip out those containing ***Calpurnia***?
  - Slow (for large corpora)
  - *NOT* ***Calpurnia*** is non-trivial
  - Other operations (e.g., find the word ***Romans*** near ***countrymen***) not feasible

Term-document Matrix

	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

Brutus *AND* ***Caesar*** but *NOT* ***Calpurnia***

1 if play contains word, 0 otherwise

Incidence vectors

- So we have a 0/1 vector for each term.
- To answer query: take the vectors for ***Brutus, Caesar*** and ***Calpurnia*** (complemented) ➔ bitwise *AND*.
- 110100 *AND* 110111 *AND* 101111 = 100100.

Bigger corpora

- Consider n = 1M documents, each with about 1K terms.
- Avg 6 bytes/term incl spaces/punctuation
  - 6GB of data in the documents.
- Say there are m = 500K *distinct* terms among these.

Can’t build the matrix

- 500K x 1M matrix has half-a-trillion 0’s and 1’s.
- But it has no more than one billion 1’s.
  - matrix is extremely sparse.
- What’s a better representation?
  - We only record the 1 positions.

Why?

Inverted index

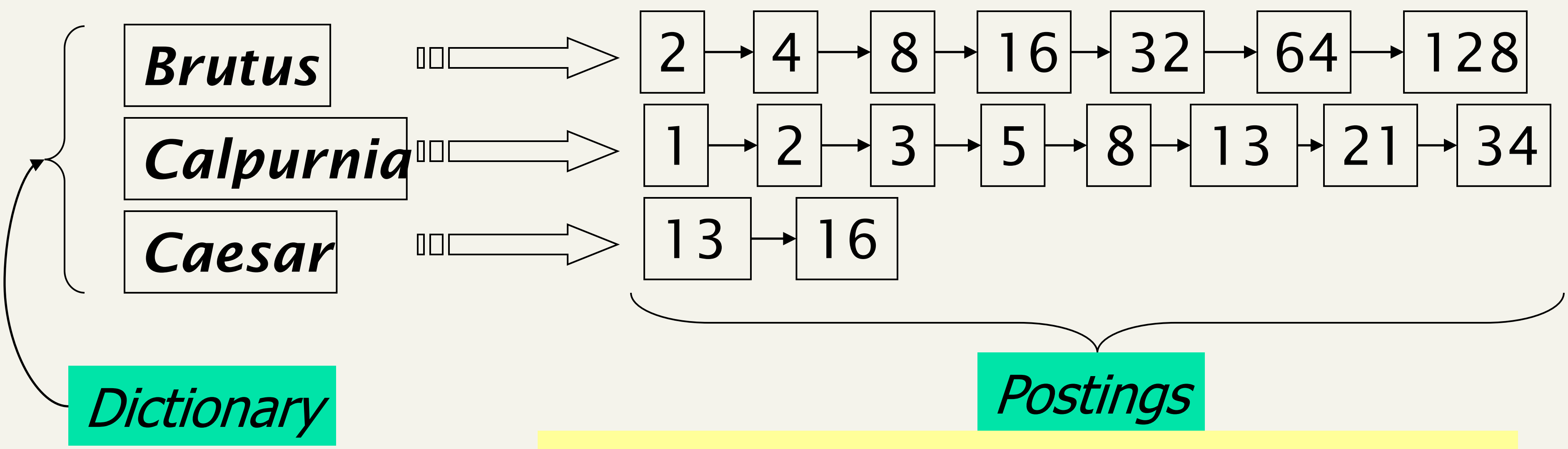
- For each term *T*, we must store a list of all documents that contain *T*.
- Do we use an array or a linked list for this?

Brutus	2	4	8	16	32	64	128	
Calpurnia	1	2	3	5	8	13	21	34
Caesar	13	16						

What happens if the word ***Caesar*** is added to document 14?

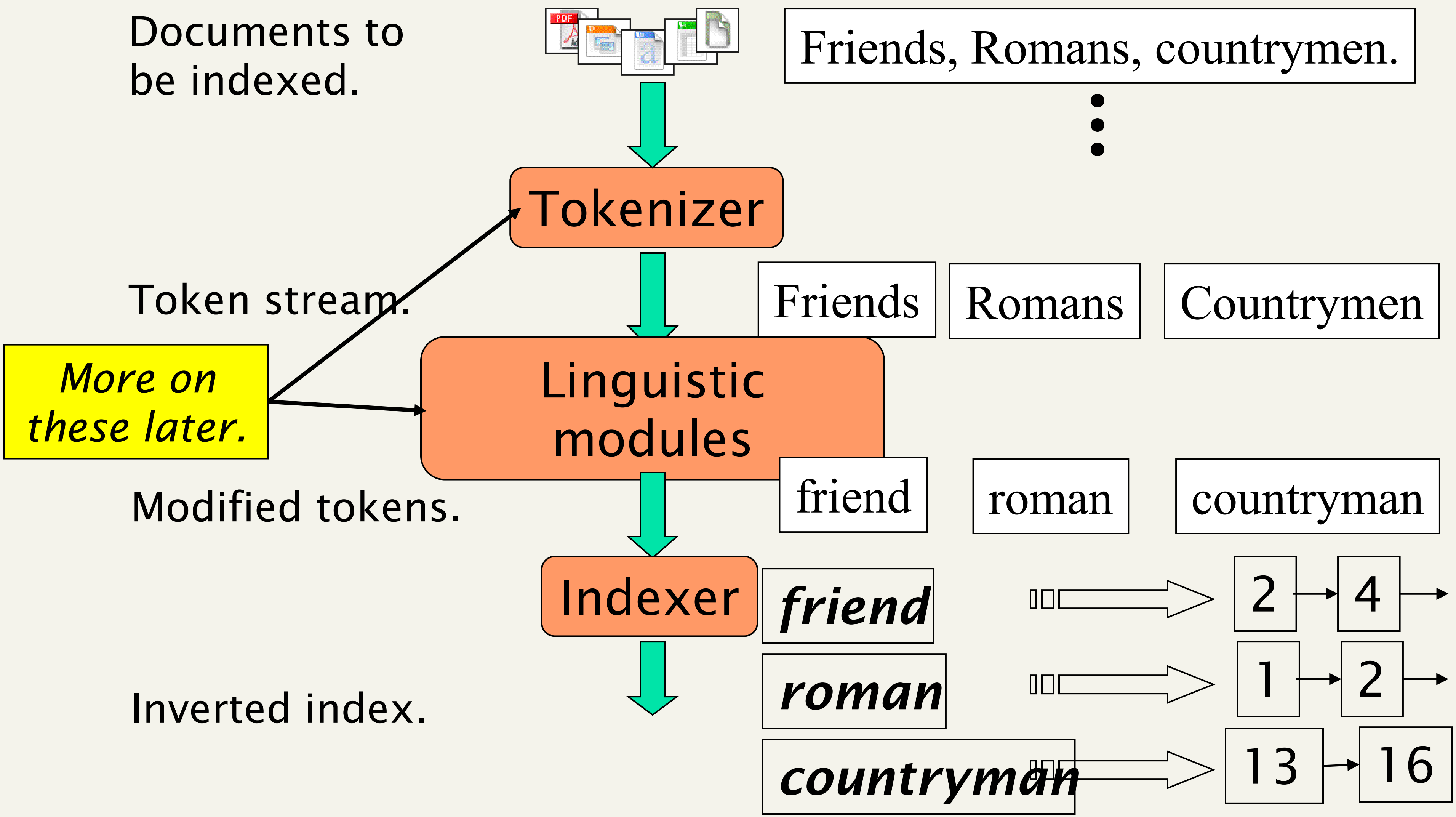
Inverted index

- Linked lists generally preferred to arrays
  - + Dynamic space allocation
  - + Insertion of terms into documents easy
  - - Space overhead of pointers



Sorted by docID (more later on why).

Inverted index construction



The index we just built

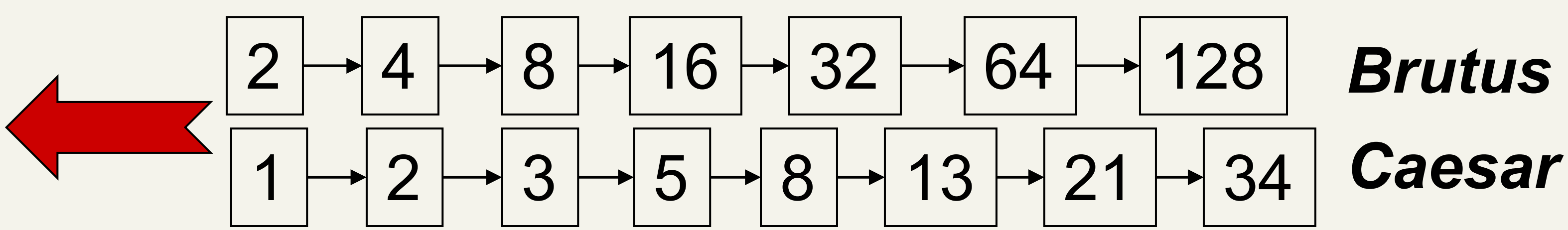
- How do we process a query?
  - Later - what kinds of queries can we process?

Query processing

- Consider processing the query:

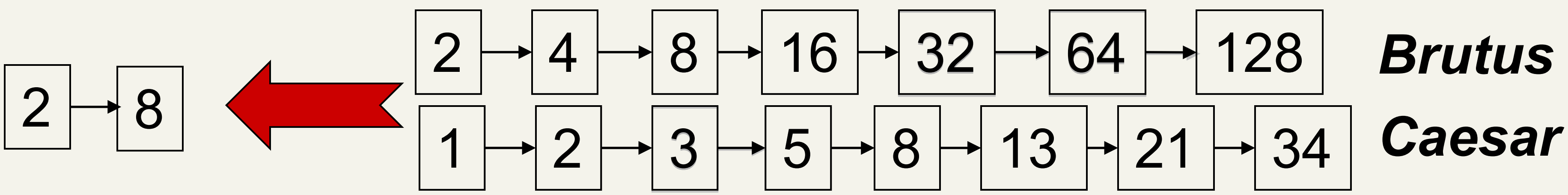
  Brutus *AND* ***Caesar***

  - Locate ***Brutus*** in the Dictionary;
    - Retrieve its postings.
  - Locate ***Caesar*** in the Dictionary;
    - Retrieve its postings.
  - "Merge" the two postings:



The merge

- Walk through the two postings simultaneously, in time linear in the total number of postings entries



If the list lengths are *x* and *y*, the merge takes O(*x*+*y*) operations.
Crucial: postings sorted by docID.

More general merges

- Exercise: Adapt the merge for the queries:

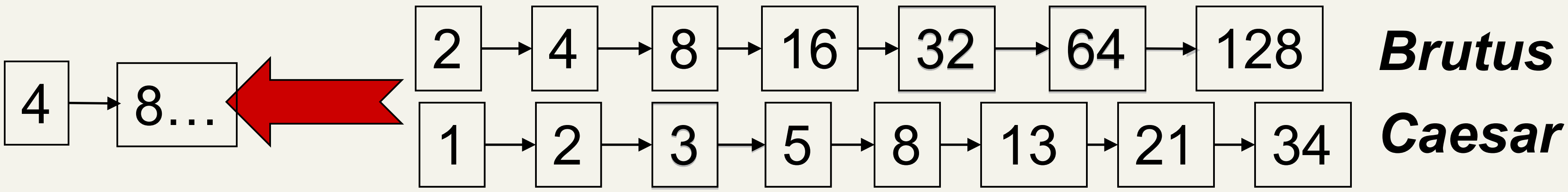
  Brutus *AND NOT* ***Caesar***

  Brutus *OR NOT* ***Caesar***

Can we still run through the merge in time O(x+y)?

The merge (Brutus and Not Caesar)

- Walk through the two postings simultaneously, in time linear in the total number of postings entries



Clustering and classification

- Given a set of docs, group them into clusters based on their contents.
- Given a set of topics, plus a new doc *D*, decide which topic(s) *D* belongs to.

Scoring: density-based

- Thus far: terms in a doc
- Obvious next idea: if a document talks about a topic *more*, then it is a better match (more similar)
- This leads to the idea of term weighting.

Term weighting

Term-document count matrices

- Consider the number of occurrences of a term in a document:
  - Bag of words model
  - Document is a vector in $\mathbb{N}^v$: a column below

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Terminology

- In saying term frequency we mean the number of occurrences of a term in a document.

Term frequency *tf*

- Long docs are favored because they're more likely to contain query terms (with higher frequency as well.)
- Can fix this to some extent by normalizing for document length
- But is raw *tf* the right measure?

Weighted term frequency: *tf*

- What is the relative importance of
  - 0 vs. 1 occurrence of a term in a doc
  - 1 vs. 2 occurrences
  - 2 vs. 3 occurrences …
- Unclear: while it seems that more is better, a lot isn't proportionally better than a few
  - Can just use raw *tf*
  - Another option commonly used in practice:

$$wf_{t,d} = 0 \text{ if } tf_{t,d} = 0,\ 1 + \log tf_{t,d} \text{ otherwise}$$

Score computation

- Score for a query q = sum tf over all terms t in q:

$$= \sum_{t \in q} tf_{t,d}$$

- [Note: 0 if no query terms in document]
- Can use *wf* instead of *tf* in the above

An example…

- Consider the ***ides of march*** query.
  - *Julius Caesar* has 5 occurrences of ***ides***
  - No other play has ***ides***
  - ***march*** occurs in over a dozen
  - All the plays contain ***of***
- By this density-based scoring measure, the top-scoring play is likely to be the one with the most ***of***'s

*Still doesn't consider term scarcity in collection (**ides** is rarer than **of**)*

Weighting should depend on the term overall

- Which of these tells you more about a doc?
  - 10 occurrences of *hernia*?
  - 10 occurrences of *the*?
- Would like to attenuate the weight of a common term
  - But what is "common"?
- Suggest looking at collection frequency (*cf*)
  - The total number of occurrences of the term in the entire collection of documents

Document frequency

- But document frequency (*df*) may be better:
- *df* = number of docs in the corpus containing the term

Word	*cf*	*df*
try	10422	8760
insurance	10440	3997

- Document/collection frequency weighting is only possible in known (static) collection.
- So how do we make use of *df* ?

tf x idf term weights

- tf x idf measure combines:
  - term frequency (*tf*)
    - or *wf*, measure of term density in a doc
  - inverse document frequency (*idf*)
    - measure of informativeness of a term: its rarity across the whole text corpus
    - could just be raw count of number of documents the term occurs in ($idf_i = 1/df_i$)
    - but by far the most commonly used version is:

$$idf_i = \log\left(\frac{n}{df_i}\right)$$

- See Kishore Papineni, NAACL 2, 2002 for theoretical justification

Summary: tf x idf (or tf.idf)

- Assign a tf.idf weight to each term *i* in each document *d*

$$w_{i,d} = tf_{i,d} \times \log(n / df_i)$$

What is the wt of a term that occurs in all of the docs?

$$tf_{i,d} = \text{frequency of term } i \text{ in document } j$$
$$n = \text{total number of documents}$$
$$df_i = \text{the number of documents that contain term } i$$

- Increases with the number of occurrences *within* a doc
- Increases with the rarity of the term *across* the whole corpus

Real-valued term-document matrices

- Function (scaling) of count of a word in a document:
  - <u>Bag of words</u> model
  - Each is a vector in $\mathbb{R}^v$
  - Here <u>log-scaled</u> *tf.idf*

Note can be >1!

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	13.1	11.4	0.0	0.0	0.0	0.0
Brutus	3.0	8.3	0.0	1.0	0.0	0.0
Caesar	2.3	2.3	0.0	0.5	0.3	0.3
Calpurnia	0.0	11.2	0.0	0.0	0.0	0.0
Cleopatra	17.7	0.0	0.0	0.0	0.0	0.0
mercy	0.5	0.0	0.7	0.9	0.9	0.3
worser	1.2	0.0	0.6	0.6	0.6	0.0

Documents as vectors

- Each doc *j* can now be viewed as a vector of *wf*×*idf* values, one component for each term
- So we have a vector space
  - terms are axes (features)
  - docs live in this space
  - even with stemming, may have 20,000+ dimensions
- (The corpus of documents gives us a matrix, which we could also view as a vector space in which words live – transposable data)

Typical process of document clustering

- Tokenizing and stemming each document
- Build tf-idf matrix
- Calculating cosine distance between each pair of documents as a measure of similarity
- Clustering the documents using a clustering method (e.g., hierarchical clustering or k-means)
- Topic modeling using techniques such as Latent Dirichlet Allocation (LDA)

- http://brandonrose.org/clustering in Python
- Scikit Learn: http://scikit-learn.org/0.15/auto_examples/document_clustering.html

Scikit Learn example

- https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
- Word2Vec