Information Retrieval (IR)

Based on slides by
Prabhakar Raghavan, Hinrich Schütze, Ray Larson

Administrativa

- Problem Set – online tomorrow, due 4/14
  - Work in pairs
- Office hours:
  - Dan: Fridays 10-11am (CSE 588) - or via email
  - Chloe: TBA
- Dan OOT next week
- Project teams & ideas also due 4/14,
  - But I expect iteration, so do the best you can.

More Project Ideas

- Information extraction from
  - wikipedia, products, reviews,
  - hiking,
  - some specialized area
- Extracting events (concerts, …)
  - E.g. time is easy, but the event, harder
- Webcam classification (location, orientation, …)
- WebTables

Main Ideas for Today

- Information retrieval
  - Boolean, similarity, relevance
- Term-document matrix
- Inverted index
- Stemming & stop-words
- Vector-space model
- TF-IDF
- Precision, recall & F measure

Query

- Which plays of Shakespeare contain the words Brutus AND Caesar but NOT Calpurnia?
- Could grep all of Shakespeare’s plays for Brutus and Caesar then strip out lines containing Calpurnia?
  - Slow (for large corpora)
  - NOT is hard to do
  - Other operations (e.g., find the Romans NEAR countrymen) unfeasibly slow

Term-document incidence

<table>
<thead>
<tr>
<th></th>
<th>Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Brutus</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>worser</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

1 if play contains word, 0 otherwise
Booleans over 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.
- \[110100 \text{ AND } 110111 \text{ AND } 101111 = 100100.\]

Answers to Query

- Antony and Cleopatra, Act III, Scene ii
  Agrippa [Aside to DOMITIUS ENOBARBUS]:
  Why, Enobarbus, When Antony found Julius Caesar dead, He cried almost to roaring; and he wept When at Philippi he found Brutus slain.

- Hamlet, Act III, Scene ii
  Lord Polonius:
  I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

So we’re done?

- Pretty easy, huh?

Bigger Corpora

- Consider
  - \(n = 1\)M documents,
  - each with about 1K terms.
- Avg 6 bytes/term incl spaces/punctuation
  - 6GB of data.
- Say there are \(m = 500K\) distinct terms....

Can’t Build the Matrix

- 500K x 1M matrix: 500 Billion 0’s and 1’s.
- But it has no more than 1 billion 1’s.
- matrix is extremely sparse.
- What’s a better representation?

Inverted index

Documents are parsed to extract words and these are saved with the document ID.

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

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

So let it be with Caesar. The Noble Brutus hath told you Caesar was ambitious.
Later, sort inverted file by terms

<table>
<thead>
<tr>
<th>Term</th>
<th>Doc #</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>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>I</td>
<td>1</td>
</tr>
</tbody>
</table>

Multiple term entries in a single document are merged and frequency information added

Issues with index we just built

- How do we process a query?
- What terms in a doc do we index?
  - All words or only “important” ones?
- Stopword list: terms that are so common that they’re ignored for indexing.
  - e.g., the, a, an, of, to …
  - language-specific.

Issues in what to index

- Cooper’s vs. Cooper vs. Cooper's.
- Full-text vs. full text vs. [full, text] vs. fulltext.
- Accents: résumé vs. resume.

Punctuation

- Ne’er: use language-specific, handcrafted “locale” to normalize.
- State-of-the-art: break up hyphenated sequence.
- U.S.A. vs. USA - use locale.
- a.out

Numbers

- 3/12/91
- Mar. 12, 1991
- 55 B.C.
- B-52
- 100.2.86.144
  - Generally, don’t index as text
  - Creation dates for docs
Case folding
- Reduce all letters to lower case
- Exception: upper case in mid-sentence
  - e.g., General Motors
  - Fed vs. fed
  - SAIL vs. sail

Thesauri and soundex
- Handle synonyms and homonyms
  - Hand-constructed equivalence classes
    - e.g., car = automobile
    - your ≠ you’re
- Index such equivalences?
- Or expand query?
  - More later ...

Spell correction
- Look for all words within (say) edit distance 3 (Insert/Delete/Replace) at query time
  - e.g., Alanis Morisette
- Spell correction is expensive and slows the query (up to a factor of 100)
  - Invoke only when index returns zero matches?
  - What if docs contain mis-spellings?

Lemmatization
- Reduce inflectional/variant forms to base form
  - E.g.,
    - am, are, is → be
    - car, cars, car’s, cars’ → car
- The boy’s cars are different colors → the boy car be different color

Stemming
- Reduce terms to their “roots” before indexing
  - Language dependent
    - E.g., automate(s), automatic, automation all reduced to automat.
  - For example, compressed and compression are both accepted as equivalent to compress.

Porter’s algorithm
- Most common algorithm for stemming English
- Conventions + 5 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 available:
  - http://www.sims.berkeley.edu/~hears/irbook/porter.html
Typical rules in Porter

- *sses* → *ss*
- *ies* → *i*
- *ational* → *ate*
- *tional* → *tion*

Challenges

- Sandy
- Sanded
  Sand ???

Beyond Term Search

- Phrases?

- Proximity: Find *Gates NEAR Microsoft*.
  - Index must capture position info in docs.

- Zones in documents: Find documents with
  *author = Ullman* AND (text contains *automata*).

Evidence accumulation

- 1 vs. 0 occurrence of a search term
- 2 vs. 1 occurrence
- 3 vs. 2 occurrences, etc.
- Need term frequency information / docs

Ranking search results

- Boolean queries give inclusion or exclusion of docs.

- Need to measure proximity from query to each doc.

- Whether docs presented to user are singletons, or a group of docs covering various aspects of the query.

Test Corpora

<table>
<thead>
<tr>
<th>Collection</th>
<th>#Doc</th>
<th>#Term</th>
<th>Ave Tokens</th>
<th>Ave (MD)</th>
<th>Mm/tok</th>
<th>2Dr Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADE</td>
<td>90</td>
<td>15</td>
<td>2</td>
<td>498</td>
<td>&gt;55,000</td>
<td></td>
</tr>
<tr>
<td>ATP</td>
<td>102</td>
<td>14</td>
<td>2</td>
<td>367</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CACM</td>
<td>324</td>
<td>64</td>
<td>2</td>
<td>46.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMU</td>
<td>1490</td>
<td>132</td>
<td>2</td>
<td>46.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gated</td>
<td>1500</td>
<td>232</td>
<td>2</td>
<td>22.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LISA</td>
<td>3872</td>
<td>35</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Medline</td>
<td>1653</td>
<td>30</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TREC</td>
<td>11,429</td>
<td>93</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRECMB</td>
<td>34,966</td>
<td>128</td>
<td>600</td>
<td>219</td>
<td>16,140</td>
<td></td>
</tr>
<tr>
<td>TRECWEB</td>
<td>22,570</td>
<td>672</td>
<td>28</td>
<td>131</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRECDB</td>
<td>741,000</td>
<td>280</td>
<td>2000</td>
<td>85,3543</td>
<td>&gt;100,000</td>
<td></td>
</tr>
</tbody>
</table>
Standard relevance benchmarks

- TREC - National Institute of Standards and Testing (NIST) has run large IR testbed for many years
- Reuters and other benchmark sets used
- "Retrieval tasks" specified
  - sometimes as queries
- Human experts mark, for each query and for each doc, “Relevant” or “Not relevant”
  - or at least for subset that some system returned

Sample TREC query

Topper: Tobacco company advertising out the young

Description: A document will generate information on what is a widely held opinion that the tobacco industry aims to young people.

Narrative: A relevant document must report on tobacco company advertising and its relation to young people. A relevant document can address either of the questions:
1. Do tobacco companies consciously target the young, or
2. Is the tobacco industry argues, is this an erroneous public perception. The "young" may be identified as youth, diabetics, alcoholics, teenagers, high school students, and college students.

Precision and recall

- **Precision**: fraction of retrieved docs that are relevant = P(relevant|retrieved)
- **Recall**: fraction of relevant docs that are retrieved = P(retrieved|relevant)

<table>
<thead>
<tr>
<th>Relevant</th>
<th>Not Relevant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retrieved</td>
<td>tp fp</td>
</tr>
<tr>
<td>Not Retrieved</td>
<td>fn tn</td>
</tr>
</tbody>
</table>

- Precision $P = \frac{tp}{tp + fp}$
- Recall $R = \frac{tp}{tp + fn}$

Precision & Recall

- Precision: $\frac{tp}{tp + fp}$
- Recall: $\frac{tp}{tp + fn}$

Proportion of selected items that are correct

% of target items that were selected

Precision-Recall curve

Shows tradeoff

- Difficulties in using precision/recall
  - Binary relevance
  - Should average over large corpus/query ensembles
  - Need human relevance judgements
  - Heavily skewed by corpus/authorship

A combined measure: F

- Combined measure that assesses this tradeoff is F measure (weighted harmonic mean):
  \[ F = \frac{1}{\frac{1}{P} + \frac{1}{R}} = \frac{\beta^2 + 1)PR}{\beta^2 P + R} \]

- People usually use balanced F measure
  - i.e., with $\beta = 1$ or $\alpha = \frac{1}{2}$
- Harmonic mean is conservative average
  - See CJ van Rijsbergen, *Information Retrieval*
### Precision-recall curves

- Evaluation of ranked results:
  - You can return any number of results ordered by similarity
  - By taking various numbers of documents (levels of recall), you can produce a precision-recall curve

### Evaluation

- There are various other measures
  - Precision at fixed recall
    - This is perhaps the most appropriate thing for web search: all people want to know is how many good matches there are in the first one or two pages of results
  - 11-point interpolated average precision
    - The standard measure in the TREC competitions: Take the precision at 11 levels of recall varying from 0 to 1 by tenths of the documents, using interpolation (the value for 0 is always interpolated!), and average them

### Ranking models in IR

- Key idea:
  - We wish to return in order the documents most likely to be useful to the searcher
  - To do this, we want to know which documents best satisfy a query
    - An obvious idea is that if a document talks about a topic more then it is a better match
    - A query should then just specify terms that are relevant to the information need, without requiring that all of them must be present
    - Document relevant if it has a lot of the terms

### Binary term presence matrices

- Record whether a document contains a word: document is binary vector in \( \{0, 1\} \)
- Idea: Query satisfaction = overlap measure:
  \[
  |X \cap Y|
  \]

<table>
<thead>
<tr>
<th></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>0</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>1</td>
<td>1</td>
<td>0</td>
<td>1</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>Calphasa</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>1</td>
</tr>
<tr>
<td>mercy</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>woman</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Many Overlap Measures

<table>
<thead>
<tr>
<th>Measure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>Q \cap D</td>
</tr>
<tr>
<td>$2 \frac{</td>
<td>Q \cap D</td>
</tr>
<tr>
<td>$\frac{</td>
<td>Q \cap D</td>
</tr>
<tr>
<td>$\frac{</td>
<td>Q \cap D</td>
</tr>
<tr>
<td>$\frac{</td>
<td>Q \cap D</td>
</tr>
</tbody>
</table>

Documents as vectors

- Each doc $j$ can be viewed as a vector of $tf$ values, one component for each term
- So we have a vector space
  - terms are axes
  - 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)

Documents in 3D Space

Assumption: Documents that are “close together” in space are similar in meaning.

The vector space model

**Query as vector:**
- Regard query as short document
- Return the docs, ranked by distance to the query
- Easy to compute, since both query & docs are vectors.
- Developed in the SMART system (Salton, c. 1970) and standardly used by TREC participants and web IR systems

Vector Representation

- Documents & Queries represented as vectors.
- Position 1 corresponds to term 1, ... position $t$ to term $t$
- The weight of the term is stored in each position
  - $D_j = w_{d_1}, w_{d_2}, ..., w_{d_t}$
  - $Q = w_{q_1}, w_{q_2}, ..., w_{q_t}$
  - $w = 0$ if a term is absent
- Vector distance measure used to rank retrieved documents

Documents in 3D Space

Documents that are close to query (measured using vector-space metric) => returned first.
Document Space has High Dimensionality

- What happens beyond 2 or 3 dimensions?
  - Similarity still has to do with the number of shared tokens.
  - More terms - harder to understand which subsets of words are shared among similar documents.
- We will look in detail at ranking methods
  - One approach to handling high dimensionality: Clustering

TF x IDF

\[ w_{tk} = tf_{tk} \times \log\left(\frac{N}{n_k}\right) \]

- \( T_k \) = term k in document D
- \( tf_{tk} \) = frequency of term \( T_k \) in document D
- \( idf_{tk} \) = inverse document frequency of term \( T_k \) in C
- \( N \) = total number of documents in the collection C
- \( n_k \) = the number of documents in C that contain \( T_k \)
- \( idf_{tk} = \log\left(\frac{N}{n_k}\right) \)

Inverse Document Frequency

- IDF provides high values for rare words and low values for common words
- \[ \begin{align*}
  \log\left(\frac{10000}{10000}\right) &= 0 \\
  \log\left(\frac{10000}{5000}\right) &= 0.301 \\
  \log\left(\frac{10000}{20}\right) &= 2.698 \\
  \log\left(\frac{10000}{1}\right) &= 4 
\end{align*} \]

TF-IDF normalization

- Normalize the term weights
  - so longer docs not given more weight (fairness)
  - force all values to fall within a certain range: \([0, 1]\)

\[ w_{tk} = \frac{tf_{tk} \times \log\left(\frac{N}{n_k}\right)}{\sqrt{\sum_{j \neq k} (tf_{tk})^2 \times \log\left(\frac{N}{n_k}\right)}} \]

Vector space similarity

(use the weights to compare the documents)

Now, the similarity of two documents is:

\[ sim(D_i, D_j) = \sum_{k=1}^{n} w_{tk} \times w_{jk} \]

This is also called the cosine, or normalized inner product.
(Normalization was done when weighting the terms.)
What’s Cosine anyway?

One of the basic trigonometric functions encountered in trigonometry. Let \( \theta \) be an angle measured counterclockwise from the x-axis along the arc of the unit circle. Then \( \cos(\theta) \) is the horizontal coordinate of the arc endpoint. As a result of this definition, the cosine function is periodic with period \( 2\pi \).

From http://mathworld.wolfram.com/Cosine.html

Computing Cosine Similarity Scores

\[ D_1 = (0.8, 0.3) \]
\[ D_2 = (0.2, 0.7) \]
\[ Q = (0.4, 0.8) \]
\[ \cos \alpha_1 = 0.74 \]
\[ \cos \alpha_2 = 0.98 \]

Computing a similarity score

Say we have query vector \( Q = (0.4, 0.8) \)
Also, document \( D_2 = (0.2, 0.7) \)
What does their similarity comparison yield?

\[
\text{sim}(Q, D_2) = \frac{(0.4 \times 0.2) + (0.8 \times 0.7)}{\sqrt{(0.4)^2 + (0.8)^2} \times \sqrt{(0.2)^2 + (0.7)^2}}
\]
\[
= \frac{0.64}{\sqrt{0.42}} = 0.98
\]

To Think About

- How does this ranking algorithm behave?
  - Make a set of hypothetical documents consisting of terms and their weights
  - Create some hypothetical queries
  - How are the documents ranked, depending on the weights of their terms and the queries’ terms?

Summary: Why use vector spaces?

- User’s query treated as a (very) short document.
- Query \( \vec{v} \) a vector in the same space as the docs.
- Easily measure each doc’s proximity to query.
- Natural measure of scores/ranking
  - No longer Boolean.
Main Ideas for Today

- Information retrieval
  - Boolean, similarity, relevance
- Term-document matrix
- Inverted index
- Stemming & stop-words
- Vector-space model
- TF-IDF
- Precision, recall & F measure