Dealing with Text Databases

- Unstructured data
- Boolean queries
  - Sparse matrix representation
  - Inverted index
- Counts vs. frequencies
- Term frequency
- tf x idf term weights
- Documents as vectors
  - Cosine similarity
  - Dimensionality reduction
  - GEMINI
- Vectors and Boolean queries
Unstructured data

- Which plays of Shakespeare contain the words *Brutus AND Caesar* but *NOT Calumnies*?  
  - (Calpurinia, third and last wife of Julius Caesar)

- One could grep all of Shakespeare’s plays for *Brutus* and *Caesar*, then strip out lines containing *Calpurnia*?

  - Slow (for large corpora)
  - *NOT Calpurnia* is non-trivial
Term-document incidence

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

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
Answers to query
110100 AND 110111 AND 101111 = 100100

- Antony and Cleopatra
- Hamlet

![Sparse matrix representation table](image)

Sparse matrix representation

- For real data matrix becomes very big
- Matrix has much, much more zeros then ones
  - matrix is extremely sparse
  - Why? Not every term (word) in every document present
  - Associative Memory :-) • Next week!!!

- What’s a better representation?
  - We only record the 1 positions
Inverted index

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

<table>
<thead>
<tr>
<th>Term</th>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brutus</td>
<td>2 4 8 16 32 64 128</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>1 2 3 5 8 13 21 34</td>
</tr>
<tr>
<td>Caesar</td>
<td>13 16</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Term</th>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brutus</td>
<td>2 4 8 16 32 64 128</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>1 2 3 5 8 13 21 34</td>
</tr>
<tr>
<td>Caesar</td>
<td>13 16</td>
</tr>
</tbody>
</table>
Inverted index construction

Documents to be indexed.

Tokenizer

Token stream.

Linguistic modules

Modified tokens.

Indexer

Inverted index

Boolean queries: Exact match

- The Boolean Retrieval model is being able to ask a query that is a Boolean expression:
  - Boolean Queries are queries using AND, OR and NOT to join query terms
  - Views each document as a set of words (terms)
  - Is precise: document matches condition or not
Exact match

- Primary commercial retrieval tool for 3 decades

- Professional searchers (e.g., lawyers) still like Boolean queries:
  - You know exactly what you’re getting.

Scoring

- Our queries have all been Boolean
- Good for expert users with precise understanding of their needs and the corpus

- Not good for (the majority of) users with poor Boolean formulation of their needs
Scoring

- We wish to return in order the documents most likely to be useful to the searcher
- How can we rank order the docs in the corpus with respect to a query?
- Assign a score – say in \([0,1]\)
  - for each doc on each query

Incidence matrices

- Recall: Document (or a zone in it) is binary vector \(X\) in \(\{0,1\}^v\)
- Query is a vector
- Score: 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>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>
Example

- On the query *ides of march*, Shakespeare’s *Julius Caesar* has a score of 3
- All other Shakespeare plays have a score of 2 (because they contain *march*) or 1
- Thus in a rank order, *Julius Caesar* would come out tops

Overlap matching

- What’s wrong with the overlap measure?
- It doesn’t consider:
  - Term frequency in document
  - Term scarcity in collection (document mention frequency)
    - *of* is more common than *ides* or *march*
  - Length of documents
Scoring: density-based

- Obvious next: idea if a document talks about a topic *more*, then it is a better match.
- This applies even when we only have a single query term.
- Document relevant if it has a lot of the terms.
- This leads to the idea of 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 \(N^v\): a column below.

\[
\begin{array}{ccccccc}
\text{Antony and Cleopatra} & \text{Julius Caesar} & \text{The Tempest} & \text{Hamlet} & \text{Othello} & \text{Macbeth} \\
\text{Antony} & 157 & 73 & 0 & 0 & 0 & 0 \\
\text{Brutus} & 4 & 157 & 0 & 1 & 0 & 0 \\
\text{Caesar} & 232 & 227 & 0 & 2 & 1 & 1 \\
\text{Calpurnia} & 0 & 10 & 0 & 0 & 0 & 0 \\
\text{Cleopatra} & 57 & 0 & 0 & 0 & 0 & 0 \\
\text{mercy} & 2 & 0 & 3 & 5 & 5 & 1 \\
\text{worse} & 2 & 0 & 1 & 1 & 1 & 0 \\
\end{array}
\]
Counts vs. frequencies

- Consider again 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 scoring measure, the top-scoring play is likely to be the one with the most *of*

Digression: terminology

- **WARNING**: In a lot of IR literature, “frequency” is used to mean “count”
  - Thus *term frequency* in IR literature is used to mean *number of occurrences* in a doc
  - Not divided by document length (which would actually make it a frequency)

- We will conform to this misnomer
  - 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
- Can fix this to some extent by normalizing for document length
- But is raw $tf$ the right measure?

Weighting 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:
    - $t=$term, $d=$document

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

- Score for a query $q = \sum_{t \in q} tf_{t,d}$
- [Note: 0 if no query terms in document]
- This score can be zone-combined
- Can use $wf$ instead of $tf$ in the above
- 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 $n$ documents
Document frequency

- But document frequency ($df$) may be better:
- $df = \text{number of docs in the corpus containing the term}$

<table>
<thead>
<tr>
<th>Word</th>
<th>$cf$</th>
<th>$df$</th>
</tr>
</thead>
<tbody>
<tr>
<td>alfa</td>
<td>10422</td>
<td>17</td>
</tr>
<tr>
<td>insurance</td>
<td>10440</td>
<td>3997</td>
</tr>
</tbody>
</table>

- Document/collection frequency weighting is only possible in known (static) collection
  - The number of documents in the entire collection of $n$ documents
- So how do we make use of $df$?

$tf \times idf$ term weights

- $tf \times idf$ measure combines:
  - term frequency ($tf$)
    - or $wf$, some measure of term density in a doc
  - inverse document frequency ($idf$)
    - measure of informativeness of a term: its rarity across the whole 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 $d$.
  \[ w_{i,d} = tf_{i,d} \times \log(n / df_i) \]
  
  - $tf_{i,d}$ = frequency of term $i$ in document $d$.
  - $n$ = total number of documents.
  - $df_i$ = 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:
  
  - Bag of words model.
  - Each is a vector in $\mathbb{R}^v$.
  - Here log-scaled $tf.idf$.

Note can be >1!

<table>
<thead>
<tr>
<th>Term</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>13.1</td>
<td>11.4</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Brutus</td>
<td>3.0</td>
<td>8.3</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Caesar</td>
<td>2.3</td>
<td>2.3</td>
<td>0.0</td>
<td>0.5</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0.0</td>
<td>11.2</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>17.7</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>mercy</td>
<td>0.5</td>
<td>0.0</td>
<td>0.7</td>
<td>0.9</td>
<td>0.9</td>
<td>0.3</td>
</tr>
<tr>
<td>worser</td>
<td>1.2</td>
<td>0.0</td>
<td>0.6</td>
<td>0.6</td>
<td>0.6</td>
<td>0.0</td>
</tr>
</tbody>
</table>
Documents as vectors

- Each doc $j$ can now be viewed as a vector of $wf \times idf$ 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

Why turn docs into vectors?

- First application: **Query-by-example**
  - Given a doc $d$, find others “like” it.

- Now that $d$ is a vector, find vectors (docs) “near” it....
Intuition

Postulate: Documents that are "close together" in the vector space talk about the same things.

Desiderata for proximity

- If $d_1$ is near $d_2$, then $d_2$ is near $d_1$
- If $d_1$ near $d_2$, and $d_2$ near $d_3$, then $d_1$ is not far from $d_3$
- No doc is closer to $d$ than $d$ itself
First cut

- Idea: Distance between $d_1$ and $d_2$ is the length of the vector $|d_1 - d_2|$.
  - Euclidean distance

- Why is this not a great idea?
  - We still haven’t dealt with the issue of length normalization
    - Short documents would be more similar to each other by virtue of length, not topic
  - However, we can implicitly normalize by looking at angles instead

Cosine similarity

- Distance between vectors $d_1$ and $d_2$ captured by the cosine of the angle $\theta$ between them.
- Note – this is similarity, not distance
  - No triangle inequality for similarity.

Variation of $\theta$ ranges from 0 to 1!!!!!
Cosine similarity

- A vector can be *normalized* (given a length of 1) by dividing each of its components by its length – here we use the $L_2$ norm
  \[ \|\mathbf{x}\|_2 = \sqrt{\sum x_i^2} \]

- This maps vectors onto the unit sphere:
- Then, \( |\tilde{d}_j| = \sqrt{\sum_{i=1}^{n} w_{i,j}} = 1 \)
- Longer documents don’t get more weight

Normalized vectors

- For normalized vectors, the cosine is simply the dot product:
  \[ \cos(\tilde{d}_j, \tilde{d}_k) = \tilde{d}_j \cdot \tilde{d}_k \]

- Varies from 0 to 1!!!!!
Example

- Docs: Austen's *Sense and Sensibility*, *Pride and Prejudice*; Bronte's *Wuthering Heights*. tf weights

<table>
<thead>
<tr>
<th></th>
<th>SaS</th>
<th>PaP</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>affection</td>
<td>115</td>
<td>58</td>
<td>20</td>
</tr>
<tr>
<td>jealous</td>
<td>10</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>gossip</td>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>SaS</th>
<th>PaP</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>affection</td>
<td>0.996</td>
<td>0.993</td>
<td>0.847</td>
</tr>
<tr>
<td>jealous</td>
<td>0.087</td>
<td>0.120</td>
<td>0.466</td>
</tr>
<tr>
<td>gossip</td>
<td>0.017</td>
<td>0.000</td>
<td>0.254</td>
</tr>
</tbody>
</table>

- $\cos(\text{SAS}, \text{PAP}) = 0.996 \times 0.993 + 0.087 \times 0.120 + 0.017 \times 0.0 = 0.999$
- $\cos(\text{SAS}, \text{WH}) = 0.996 \times 0.847 + 0.087 \times 0.466 + 0.017 \times 0.254 = 0.889$

- Euclidean distance between vectors:

$$|d_j - d_k| = \sqrt{\sum_{i=1}^{n} (d_{i,j} - d_{i,k})^2}$$

- For normalized vectors, Euclidean distance gives the same proximity ordering as the cosine measure
Queries in the vector space model

**Central idea: the query as a vector:**
- We regard the query as short document
- We return the documents ranked by the closeness of their vectors to the query, also represented as a vector

\[
sim(d_j, d_q) = \frac{\tilde{d}_j \cdot \tilde{d}_q}{\|\tilde{d}_j\| \|\tilde{d}_q\|} = \frac{\sum_{i=1}^{n} w_{i,j} w_{i,q}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2} \sqrt{\sum_{i=1}^{n} w_{i,q}^2}}
\]

- Note that \(d_q\) is very sparse!
- **Varies from 0 to 1!!!!!!**

Dimensionality reduction

- What if we could take our vectors and “pack” them into fewer dimensions (say 50,000→100) while preserving distances?

- (Well, almost.)
  - Speeds up cosine computations
Normalized vectors

For normalized vectors, the cosine is simply the dot product:

$$\cos(\vec{d}_j, \vec{d}_k) = \vec{d}_j \cdot \vec{d}_k$$

Varies from 0 to 1!!!!!
Can apply GEMINI?

- Our projection defines a hierarchy of subspace
- Lower bounding lemma is valid, why?
- Similarity becomes larger, with additional dimensions....
- **Varies from 0 to 1!!!!!!**

- \[ \cos(\text{SAS, PAP}) = .996 \times .993 + .087 \times .120 + .017 \times 0.0 = 0.999 \]

Random projection onto \( k<<m \) axes

- Choose a random direction \( x_1 \) in the vector space.
- For \( i = 2 \) to \( k \),
  - Choose a random direction \( x_i \) that is orthogonal to \( x_1, x_2, \ldots, x_{i-1} \)
  - Project each document vector into the subspace spanned by \( \{x_1, x_2, \ldots, x_k\} \)
E.g., from 3 to 2 dimensions

$x_1$ is a random direction in $(t_1,t_2,t_3)$ space.
$x_2$ is chosen randomly but orthogonal to $x_1$.

Lower bounding lemma

- $\text{sim}_{\text{projection}}(F(O_1), F(O_2)) \geq \text{sim}(O_1, O_2) \geq \epsilon$?

- Lower bounding lemma is not valid
- Why, the bigger the similarity, the nearer the objects to each other
That means that we are not guaranteed to have selected all the objects we wanted plus some additional false hits in the feature space.....

It means however, that if an object is similar enough in the lower dimensional space, we do not need look further on!!

We have an upper bound lemma.....

\[
\text{sim}_{\text{projection}}(F(O_1), F(O_2)) \leq \text{sim}(O_1, O_2) \leq \varepsilon
\]

Similarity becomes larger, with additional dimensions....

Vectors and Boolean queries

- Vectors and Boolean queries really don’t work together very well
- In the space of terms, vector proximity selects by spheres: e.g., all docs having cosine similarity \(\geq 0.5\) to the query
- Boolean queries on the other hand, select by (hyper-)rectangles and their unions/intersections
- Round peg - square hole
- Unstructured data
- Boolean queries
  - Sparse matrix representation
  - Inverted index
- Counts vs. frequencies
- Term frequency
- tf x idf term weights
- Documents as vectors
  - Cosine similarity
  - Dimensionality reduction
  - GEMINI
- Vectors and Boolean queries