Clarification of TF-IDF score

• Some of the slides show this formula:

\[
\text{tfidf}(t, d) = (1 + \log(tf_{t,d})) \times \log\left(\frac{|\text{corpus}|}{df_{t,d}}\right)
\]

• Precisely it should be:

\[
\text{tfidf}(t, d) = \text{WTF}(t, d) \times \log\left(\frac{|\text{corpus}|}{df_{t,d}}\right)
\]

• The difference is just the special case when \( tf = 0 \)

\[
\text{WTF}(t, d)
1 \quad \text{if } tf_{t,d} = 0
2 \quad \text{then } return(0)
3 \quad \text{else } return(1 + \log(tf_{t,d}))
\]
Queries in the vector space model

- Central idea: the query is a vector
- We regard the query as a short document
- We return the documents ranked by the closeness of their vectors to the query (also a vector)

$$sim(q, d_i) = \frac{\vec{V}(q) \cdot \vec{V}(d_i)}{|\vec{V}(q)||\vec{V}(d_i)|}$$

- Note that q is very sparse!
Vector Space Scoring

Cosine Similarity Score

- Also called cosine similarity

\[
\mathbf{V}(d_1) \cdot \mathbf{V}(d_2) = \frac{||\mathbf{V}(d_1)||||\mathbf{V}(d_2)||}{\cos(\theta)}
\]

\[
\cos(\theta) = \frac{\mathbf{V}(d_1) \cdot \mathbf{V}(d_2)}{||\mathbf{V}(d_1)||||\mathbf{V}(d_2)||}
\]

\[
sim(d_1, d_2) = \frac{\mathbf{V}(d_1) \cdot \mathbf{V}(d_2)}{||\mathbf{V}(d_1)||||\mathbf{V}(d_2)||}
\]
Cosine Similarity Score

- Define: dot product

\[ \vec{V}(d_1) \cdot \vec{V}(d_2) = \sum_{i=t_1}^{t_n} (\vec{V}(d_1)_i \vec{V}(d_2)_i) \]

\[ \vec{V}(d_1) \cdot \vec{V}(d_2) = (13.1 \times 11.4) + (3.0 \times 8.3) + (2.3 \times 2.3) + (0 \times 11.2) + (17.7 \times 0) + (0.5 \times 0) + (1.2 \times 0) \]

\[ = 179.53 \]
Vector Space Scoring

Cosine Similarity Score

- Define: **Euclidean Length**

\[
|\vec{V}(d_1)| = \sqrt{\sum_{i=t_1}^{t_n} (\vec{V}(d_1)_i \vec{V}(d_1)_i)}
\]

\[
|\vec{V}(d_1)| = \sqrt{(13.1 \times 13.1) + (3.0 \times 3.0) + (2.3 \times 2.3) + (17.7 \times 17.7) + (0.5 \times 0.5) + (1.2 \times 1.2)} = 22.38
\]
**Cosine Similarity Score**

- **Define:** Euclidean Length

\[
|\vec{V}(d_1)| = \sqrt{\sum_{i=t_1}^{t_n} (\vec{V}(d_1)_i \vec{V}(d_1)_i)}
\]

<table>
<thead>
<tr>
<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>
</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>
</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>
</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>
</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>
</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>
</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>
</tr>
</tbody>
</table>

\[
|\vec{V}(d_1)| = \sqrt{(11.4 \times 11.4) + (8.3 \times 8.3) + (2.3 \times 2.3) + (11.2 \times 11.2)}
\]

= 18.15
Cosine Similarity Score

- Example

\[ sim(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)||\vec{V}(d_2)|} \]

\[ = \frac{179.53}{22.38 \times 18.15} \]

\[ = 0.442 \]
Exercise

• Rank the following by decreasing cosine similarity.
  • Assume tf-idf weighting:
    • Two docs that have only frequent words in common
      • (the, a, an, of)
    • Two docs that have no words in common
    • Two docs that have many rare words in common
      • (mocha, volatile, organic, shade-grown)
Spamming indices

- This was invented before spam
- Consider:
  - Indexing a sensible passive document collection
  - vs.
  - Indexing an active document collection, where people, companies, bots are shaping documents to maximize scores
- Vector space scoring may not be as useful in this context.
Vector Space Scoring

Interaction: vectors and phrases

- Scoring phrases doesn’t naturally fit into the vector space world:
  - How do we get beyond the “bag of words”?
  - “dark roast” and “pot roast”
  - There is no information on “dark roast” as a phrase in our indices.

- Biword index can treat some phrases as terms
  - postings for phrases
  - document wide statistics for phrases
Interaction: vectors and phrases

• Theoretical problem:
  • Axes of our term space are now correlated
  • There is a lot of shared information in “light roast” and “dark roast” rows of our index

• End-user problem:
  • A user doesn’t know which phrases are indexed and can more effectively discriminate results.
Multiple queries for phrases and vectors

- Query: “rising interest rates”
- Iterative refinement:
  - Run the phrase query vector with 3 words as a term.
  - If not enough results, run 2-phrase queries and fold into results: “rising interest” “interest rates”
  - If still not enough results run query with three words as separate terms.
Vectors and Boolean queries

- Ranked queries and Boolean queries don’t work very well together
- In term space
  - ranked queries select based on sector containment - cosine similarity
  - boolean queries select based on rectangle unions and intersections
Vector Space Scoring

Vectors and wild cards

• How could we work with the query, “quick* print*”?
• Can we view this as a bag of words?
• What about expanding each wild-card into the matching set of dictionary terms?
• Danger: Unlike the boolean case, we now have tfs and idfs to deal with
• Overall, not a great idea
Vectors and other operators

• Vector space queries are good for no-syntax, bag-of-words queries
  • Nice mathematical formalism
  • Clear metaphor for similar document queries
  • Doesn’t work well with Boolean, wild-card or positional query operators
• But ...
Query language vs. Scoring

- Interfaces to the rescue
- Free text queries are often separated from operator query language
- Default is free text query
- Advanced query operators are available in “advanced query” section of interface
- Or embedded in free text query with special syntax
  - aka -term -“terma termb”
Alternatives to tf-idf

- Sublinear tf scaling
  - 20 occurrences of “mole” does not indicate 20 times the relevance
  - This motivated the WTF score.
    \[
    \text{WTF}(t, d) = \begin{cases} 
    0 & \text{if } tf_{t,d} = 0 \\
    \text{return}(1 + \log(tf_{t,d})) & \text{else}
    \end{cases}
    \]

- There are other variants for reducing the impact of repeated terms
TF Normalization

- Normalize tf weights by maximum tf in that document

\[ ntf_{t,d} = \alpha + (1 - \alpha) \frac{tf_{t,d}}{tf_{max}(d)} \]

- \( \alpha \) is a smoothing term from (0 - 1.0) ~0.4 in practice
- This addresses a length bias.
- Take one document, repeat it, WTF goes up
TF Normalization

- Normalize tf weights by maximum tf in that document

\[ ntf_{t,d} = \alpha + (1 - \alpha) \frac{tf_{t,d}}{tf_{\text{max}}(d)} \]

- A change in the stop word list can change weights drastically - hard to tune
- Still based on bag of words model
- One outlier word, repeated many times might throw off the algorithmic understanding of the content
### Term Frequency

<table>
<thead>
<tr>
<th>Term</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)atural</td>
<td>tf&lt;sub&gt;t,d&lt;/sub&gt;</td>
</tr>
<tr>
<td>(l)ogarithm</td>
<td>1 + log(tf&lt;sub&gt;t,d&lt;/sub&gt;)</td>
</tr>
<tr>
<td>(a)ugmented</td>
<td>α + (1 − α) tf&lt;sub&gt;t,d&lt;/sub&gt;/tf&lt;sub&gt;max&lt;/sub&gt;(d)</td>
</tr>
<tr>
<td>(b)oolean</td>
<td>tf&lt;sub&gt;t,d&lt;/sub&gt; &gt; 0? 1 : 0</td>
</tr>
<tr>
<td>(L)ogaverage</td>
<td>1 + log(tf&lt;sub&gt;t,d&lt;/sub&gt;) / (1 + log(average&lt;sub&gt;t∈d&lt;/sub&gt;(tf&lt;sub&gt;t,d&lt;/sub&gt;)))</td>
</tr>
</tbody>
</table>

### Document Frequency

<table>
<thead>
<tr>
<th>Document</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>(n)one</td>
<td>1</td>
</tr>
<tr>
<td>(t)idf</td>
<td>log</td>
</tr>
<tr>
<td>(p)robidf</td>
<td>max{0, log (</td>
</tr>
<tr>
<td>(c)osine</td>
<td>1/ (√w&lt;sub&gt;1&lt;/sub&gt; + w&lt;sub&gt;2&lt;/sub&gt; + ... + w&lt;sub&gt;m&lt;/sub&gt;)</td>
</tr>
<tr>
<td>(u)pivoted</td>
<td>1/u</td>
</tr>
<tr>
<td>(b)yte</td>
<td>1/CharLength&lt;sup&gt;α&lt;/sup&gt;, α &lt; 1</td>
</tr>
</tbody>
</table>

### Normalization

- SMART system of describing your IR vector algorithm
- `ddd.qqq` (ddd = document weighting) (qqq = query weighting)
- first is term weighting, second is document, then normalization
- lnc.ltc is what?