Heaps’ law

Vocabulary size $M$ as a function of collection size $T$ (number of tokens) for Reuters-RCV1. For these data, the dashed line $\log_{10} M = 0.49 + \log_{10} T + 1.64$ is the best least squares fit. Thus, $M = 10^{1.64} T^{0.49}$ and $k = 10^{1.64} \approx 44$ and $b = 0.49$. 
Zipf’s law

- Zipf’s law: The $i^{th}$ most frequent term has frequency proportional to $1/i$.
- $c_f \propto \frac{1}{i}$
- $c_f$ is collection frequency: the number of occurrences of the term in the collection.
- So if the most frequent term ($the$) occurs $c_f_1$ times, then the second most frequent term ($of$) has $c_f_1/2$ occurrences, ...
- ... the third most frequent term ($and$) has $c_f_1/3$ occurrences etc.
- About half of all vocabulary terms occur only once in the collection. (hapax legomena)
- Zipf’s law is an example of a power law.

Variable byte (VB) code

- Dedicate 1 bit (high bit) to be a continuation bit $c$.
- If the gap $G$ fits within 7 bits, binary-encode it in the 7 available bits and set $c = 1$.
- Else: set $c = 0$, encode high-order 7 bits and then use one or more additional bytes to encode the lower order bits using the same algorithm.

Gamma codes for gap encoding

- You can get even more compression with bitlevel code.
- Gamma code is the best known of these.
- Represent a gap $G$ as a pair of length and offset.
- Offset is the gap in binary, with the leading bit chopped off.
- For example 13 → 1101 → 101
- Length is the length of offset.
- For 13 (offset 101), this is 3.
- Encode length in unary code: 1110.
- Gamma code of 13 is the concatenation of length and offset: 1110101.
Problem with Boolean search: Feast or famine

- Boolean queries often result in either too few (=0) or too many (1000s) results.
- Query 1: “standard user dlink 650” → 200,000 hits
- Query 2: “standard user dlink 650 no card found”: 0 hits
- It takes a lot of skill to come up with a query that produces a manageable number of hits.
- With a ranked list of documents it does not matter how large the retrieved set is.

Scoring as the basis of ranked retrieval

- We wish to return in order the documents most likely to be useful to the searcher.
- How can we rank-order the documents in the collection with respect to a query?
- Assign a score – say in $[0, 1]$ – to each document
- This score measures how well document and query “match”.

Ranked retrieval

- Thus far, our queries have all been Boolean.
  - Documents either match or don’t.
- Good for expert users with precise understanding of their needs and the collection.
- Also good for applications: Applications can easily consume 1000s of results.
- Not good for the majority of users.
- Most users are not capable of writing Boolean queries (or they are, but they think it’s too much work).
- Most users don’t want to wade through 1000s of results.
- This is particularly true of web search.
Query-document matching scores

- We need a way of assigning a score to a query/document pair.
- Let's start with a one-term query.
- If the query term does not occur in the document: score should be 0.
- The more frequent the query term in the document, the higher the score.
- We will look at a number of alternatives for doing this.

Take 1: Jaccard coefficient

- Recall from IIR 3: A commonly used measure of overlap of two sets.
- Let \( A \) and \( B \) be two sets.
- Jaccard coefficient:
  \[
  \text{JACCARD}(A, B) = \frac{|A \cap B|}{|A \cup B|}
  \]
- \( \text{JACCARD}(A, A) = 1 \)
- \( \text{JACCARD}(A, B) = 0 \) if \( A \cap B = 0 \)
- \( A \) and \( B \) don't have to be the same size.
- Always assigns a number between 0 and 1.

Jaccard coefficient: Example

- What is the query-document match score that the Jaccard coefficient computes for:
  - Query: "ides of March"
  - Document "Caesar died in March"
- ?

What's wrong with Jaccard?

- It doesn't consider term frequency (how many occurrences a term has).
- Rare terms are more informative than frequent terms. Jaccard doesn't consider this information.
- We need a more sophisticated way of normalizing for length.
- Later in this lecture, we'll use \( |A \cap B|/\sqrt{|A \cup B|} \) (cosine) . . .
- . . . instead of \( |A \cap B|/|A \cup B| \) (Jaccard) for length normalization.
Recall: Binary incidence matrix

<table>
<thead>
<tr>
<th>Characters</th>
<th>Anthony</th>
<th>Julius</th>
<th>Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anthony</td>
<td>1</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>0</td>
<td>1</td>
<td>0</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>
<td>0</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1</td>
<td>0</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>
<td>0</td>
</tr>
<tr>
<td>Mercy</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Worser</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

... Each document is represented by a binary vector $\in \{0, 1\}^{|V|}$.

From now on, we will use the frequencies of terms

<table>
<thead>
<tr>
<th>Characters</th>
<th>Anthony</th>
<th>Julius</th>
<th>Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anthony</td>
<td>157</td>
<td>73</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Brutus</td>
<td>4</td>
<td>157</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>232</td>
<td>227</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>57</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Mercy</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td>5</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>Worser</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

... Each document is represented by a count vector $\in \mathbb{N}^{|V|}$.

Bag of words model

- We do not consider the order of words in a document.
- *John is quicker than Mary* and *Mary is quicker than John* are represented the same way.
- This is called a **bag of words model**.
- In a sense, this is a step back: The positional index was able to distinguish these two documents.
- We will look at “recovering” positional information later in this course.
- For now: bag of words model

Term frequency $tf$

- The term frequency $tf_{t,d}$ of term $t$ in document $d$ is defined as the **number of times that $t$ occurs in $d$**.
- We want to use $tf$ when computing query-document match scores.
- But how?
- Raw term frequency is not what we want.
- A document with 10 occurrences of the term is **more relevant than a document with one occurrence of the term**.
- But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.
Log frequency weighting

- The log frequency weight of term $t$ in $d$ is defined as follows:
  $$w_{t,d} = \begin{cases} 
1 + \log_{10} t_{f,t,d} & \text{if } t_{f,t,d} > 0 \\
0 & \text{otherwise}
\end{cases}$$

- $0 \rightarrow 0$, $1 \rightarrow 1$, $2 \rightarrow 1.3$, $10 \rightarrow 2$, $1000 \rightarrow 4$, etc.
- Score for a document-query pair: sum over terms $t$ in both $q$ and $d$:
  $$\text{matching-score} = \sum_{t \in q \cap d} (1 + \log t_{f,t,d})$$
- The score is 0 if none of the query terms is present in the document.

Document frequency

- Rare terms are more informative than frequent terms.
- Consider a term in the query that is rare in the collection (e.g., ARACHNOCENTRIC)
  - A document containing this term is very likely to be relevant.
  - We want a high weight for rare terms like ARACHNOCENTRIC.
- Consider a term in the query that is frequent in the collection (e.g., HIGH, INCREASE, LINE)
  - A document containing this term is more likely to be relevant than a document that doesn’t. But it’s not a sure indicator of relevance.
  - For frequent terms, we want positive weights for words like HIGH, INCREASE, and LINE, but lower weights than for rare terms.
- We will use document frequency to factor this into computing the matching score.
- The document frequency is the number of documents in the collection that the term occurs in.

idf weight

- $d_{f,t}$ is the document frequency, the number of documents that $t$ occurs in.
- $d_{f}$ is an inverse measure of the informativeness of the term.
- We define the idf weight of term $t$ as follows:
  $$\text{idf}_t = \log_{10} \frac{N}{d_{f,t}}$$
- idf is a measure of the informativeness of the term.
- We use $\log N/d_{f,t}$ instead of $N/d_{f,t}$ to “dampen” the effect of idf.
- So we use the log transformation for both term frequency and document frequency.
Examples for idf

Compute \( \text{idf}_t \) using the formula:
\[
\text{idf}_t = \log_{10} \frac{1,000,000}{\text{df}_t}
\]

<table>
<thead>
<tr>
<th>term</th>
<th>( \text{df}_t )</th>
<th>( \text{idf}_t )</th>
</tr>
</thead>
<tbody>
<tr>
<td>calpurnia</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>animal</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>sunday</td>
<td>1000</td>
<td>3</td>
</tr>
<tr>
<td>fly</td>
<td>10,000</td>
<td>2</td>
</tr>
<tr>
<td>under</td>
<td>100,000</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1,000,000</td>
<td>0</td>
</tr>
</tbody>
</table>

Effect of idf on ranking

- \( \text{idf} \) affects the ranking of documents only if the query has at least two terms.
- For example, in the query “arachnocentric line”, \( \text{idf} \) weighting increases the relative weight of \text{ARACHNOCENTRIC} and decreases the relative weight of \text{LINE}.
- \( \text{idf} \) has no effect on ranking for one-term queries.
- Questions about \( \text{idf} \)?

Collection frequency vs. Document frequency

<table>
<thead>
<tr>
<th>Word</th>
<th>Collection frequency</th>
<th>Document frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSURANCE</td>
<td>10440</td>
<td>3997</td>
</tr>
<tr>
<td>TRY</td>
<td>10422</td>
<td>8760</td>
</tr>
</tbody>
</table>

The collection frequency of \( t \) is the number of tokens of \( t \) in the collection where we count multiple occurrences.

- Why these numbers?
- Which word is a better search term (and should get a higher weight)?
- This example suggests that \( \text{df} \) is better for weighting that \( \text{cf} \).

tf-idf weighting

- The tf-idf weight of a term is the product of its \( \text{tf} \) weight and its \( \text{idf} \) weight.

\[
\text{wt}, d = (1 + \log \text{tf}_{t,d}) \cdot \log \frac{N}{\text{df}_t}
\]

- Best known weighting scheme in information retrieval
- Note: the “-” in tf-idf is a hyphen, not a minus sign!
- Alternative names: \( \text{tf.idf} \), \( \text{tf} \times \text{idf} \)
Summary: tf-idf

Assign a tf-idf weight for each term \( t \) in each document \( d \):
\[
    w_{t,d} = (1 + \log tf_{t,d}) \cdot \log \frac{N}{df_t}
\]
- \( N \): total number of documents
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Term, collection and document frequency

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>term frequency</td>
<td>( tf_{t,d} )</td>
<td>number of occurrences of ( t ) in ( d )</td>
</tr>
<tr>
<td>document frequency</td>
<td>( df_t )</td>
<td>number of documents in the collection that ( t ) occurs in</td>
</tr>
<tr>
<td>collection frequency</td>
<td>( cf_t )</td>
<td>total number of occurrences of ( t ) in the collection</td>
</tr>
</tbody>
</table>

- Relationship between \( df \) and \( cf \)?
- Relationship between \( tf \) and \( cf \)?

Outline

1 Recap
2 Term frequency
3 tf-idf weighting
4 The vector space

Binary → count → weight matrix

<table>
<thead>
<tr>
<th></th>
<th>Anthony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANTHONY</td>
<td>5.25</td>
<td>3.18</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.35</td>
<td></td>
</tr>
<tr>
<td>BRUTUS</td>
<td>1.21</td>
<td>6.10</td>
<td>0.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>CAESAR</td>
<td>8.59</td>
<td>2.54</td>
<td>0.0</td>
<td>1.51</td>
<td>0.25</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>CALPURNIA</td>
<td>0.0</td>
<td>1.54</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>CLEOPATRA</td>
<td>2.85</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td></td>
</tr>
<tr>
<td>MERCY</td>
<td>1.51</td>
<td>0.0</td>
<td>1.90</td>
<td>0.12</td>
<td>5.25</td>
<td>0.88</td>
<td></td>
</tr>
<tr>
<td>WORSEER</td>
<td>1.37</td>
<td>0.0</td>
<td>0.11</td>
<td>4.15</td>
<td>0.25</td>
<td>1.95</td>
<td></td>
</tr>
</tbody>
</table>

... Each document is now represented by a real-valued vector of tf-idf weights \( \in \mathbb{R}^{\mid V \mid} \).
Documents as vectors

- Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$.
- So we have a $|V|$-dimensional real-valued vector space.
- Terms are axes of the space.
- Documents are points or vectors in this space.
- Very high-dimensional: tens of millions of dimensions when you apply this to a web search engine
- This is a very sparse vector - most entries are zero.

Queries as vectors

- Key idea 1: do the same for queries: represent them as vectors in the space
- Key idea 2: Rank documents according to their proximity to the query
- Proximity $\approx$ negative distance
- Recall: We’re doing this because we want to get away from the you’re-either-in-or-out Boolean model.
- Instead: rank more relevant documents higher than less relevant documents

How do we formalize vector space similarity?

- First cut: distance between two points
- ( = distance between the end points of the two vectors)
- Euclidean distance?
- Euclidean distance is a bad idea . . .
- . . . because Euclidean distance is large for vectors of different lengths.

Why distance is a bad idea

The Euclidean distance of $\vec{q}$ and $\vec{d}_2$ is large although the distribution of terms in the query $q$ and the distribution of terms in the document $d_2$ are very similar. Questions about basic vector space setup?
Use angle instead of distance

- Rank documents according to angle with query
- Thought experiment: take a document $d$ and append it to itself. Call this document $d'$.
- “Semantically” $d$ and $d'$ have the same content.
- The angle between the two documents is $0$, corresponding to maximal similarity.
- The Euclidean distance between the two documents can be quite large.

From angles to cosines

- The following two notions are equivalent.
  - Rank documents according to the angle between query and document in decreasing order
  - Rank documents according to $\cosine(query, document)$ in increasing order
- Cosine is a monotonically decreasing function of the angle for the interval $[0^\circ, 180^\circ]$

Cosine

What about angles $> 180^\circ$?
Length normalization

How do we compute the cosine?

A vector can be (length-) normalized by dividing each of its components by its length – here we use the $L_2$ norm:

$$||x||_2 = \sqrt{\sum_i x_i^2}$$

This maps vectors onto the unit sphere . . .

. . . since after normalization: $||x||_2 = \sqrt{\sum_i x_i^2} = 1.0$

As a result, longer documents and shorter documents have weights of the same order of magnitude.

Effect on the two documents $d$ and $d'$ ($d$ appended to itself) from earlier slide: they have identical vectors after length-normalization.

Cosine for normalized vectors

For normalized vectors, the cosine is equivalent to the dot product or scalar product.

$$\cos(\vec{q}, \vec{d}) = \vec{q} \cdot \vec{d} = \sum_i q_i d_i$$ (if $\vec{q}$ and $\vec{d}$ are length-normalized).

Cosine similarity between query and document

$$\cos(\vec{q}, \vec{d}) = \operatorname{SIM}(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}| |\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

$q_i$ is the tf-idf weight of term $i$ in the query.

$d_i$ is the tf-idf weight of term $i$ in the document.

$|\vec{q}|$ and $|\vec{d}|$ are the lengths of $\vec{q}$ and $\vec{d}$.

This is the cosine similarity of $\vec{q}$ and $\vec{d}$ . . . . . . or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

Cosine similarity illustrated
Cosine: Example

How similar are the novels? SaS: Sense and Sensibility, PaP: Pride and Prejudice, and WH: Wuthering Heights?

<table>
<thead>
<tr>
<th>Term</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>
<tr>
<td>wuthering</td>
<td>0</td>
<td>0</td>
<td>38</td>
</tr>
</tbody>
</table>

(To simplify this example, we don't do idf weighting.)

**Cosine: Example**

<table>
<thead>
<tr>
<th>term frequencies (counts)</th>
<th>log frequency weighting</th>
</tr>
</thead>
<tbody>
<tr>
<td>term</td>
<td>SaS</td>
</tr>
<tr>
<td>affection</td>
<td>115</td>
</tr>
<tr>
<td>jealous</td>
<td>10</td>
</tr>
<tr>
<td>gossip</td>
<td>2</td>
</tr>
<tr>
<td>wuthering</td>
<td>0</td>
</tr>
</tbody>
</table>

Computing the cosine score

**CosineScore(q)**

1. float Scores[N] = 0
2. float Length[N]
3. for each query term t do calculate $w_{t,q}$ and fetch postings list for t
4. for each pair $(d, tf_{t,d})$ in postings list do $Scores[d] += w_{t,d} \times w_{t,q}$
5. Read the array Length
6. for each $d$ do $Scores[d] = Scores[d] / Length[d]$
7. return Top K components of Scores[]
### Components of tf-idf weighting

<table>
<thead>
<tr>
<th>Term frequency</th>
<th>Document frequency</th>
<th>Normalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>n (natural) tf&lt;sub&gt;tr,d&lt;/sub&gt;</td>
<td>n (no) 1</td>
<td>n (none) 1</td>
</tr>
<tr>
<td>l (logarithm) 1 + log(tf&lt;sub&gt;tr,d&lt;/sub&gt;)</td>
<td>t (idf) log&lt;sub&gt;N(df&lt;sub&gt;t&lt;/sub&gt;)&lt;/sub&gt;</td>
<td>c (cosine) 1 / \sqrt{w_1^2 + \ldots + w_M^2}</td>
</tr>
<tr>
<td>a (augmented) 0.5 + \frac{0.5 \times tf&lt;sub&gt;tr,d&lt;/sub&gt;}{\text{max}(tf&lt;sub&gt;tr,d&lt;/sub&gt;)}</td>
<td>p (prob idf) \max(0, \log \frac{N}{df&lt;sub&gt;t&lt;/sub&gt;})</td>
<td>u (pivoted unique) 1 / u</td>
</tr>
<tr>
<td>b (boolean) \begin{cases} 1 \text{ if } tf&lt;sub&gt;tr,d&lt;/sub&gt; &gt; 0 \ 0 \text{ otherwise} \end{cases}</td>
<td>b (byte size) 1 / \text{CharLength}^\alpha, \alpha &lt; 1</td>
<td></td>
</tr>
<tr>
<td>L (log ave) \frac{1 + \log(tf&lt;sub&gt;tr,d&lt;/sub&gt;)}{\text{max}(1+\log(tf&lt;sub&gt;tr,d&lt;/sub&gt;))}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Best known combination of weighting options: Default: no weighting

---

**tf-idf example: ltn.lnc**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tf-raw tf-wght df idf weight</td>
<td>tf-raw tf-wght weight n’lized product</td>
<td></td>
<td>tf-raw tf-wght df idf weight</td>
<td>tf-raw tf-wght weight n’lized product</td>
</tr>
<tr>
<td>auto</td>
<td>0 0 5000 2.3 0</td>
<td>1 1 1 0.52 0</td>
<td>best</td>
<td>1 1 50000 1.3 1.3 0 0 0 0</td>
<td>0.04</td>
</tr>
<tr>
<td>car</td>
<td>1 1 10000 2.0 2.0</td>
<td>1 1 1 0.52 0</td>
<td>insurance</td>
<td>1 1 10000 3.0 3.0 2 1.3 1.3 0.68 2.04</td>
<td></td>
</tr>
</tbody>
</table>

Key to columns: 
- **tf-raw**: raw (unweighted) term frequency
- **tf-wght**: logarithmically weighted term frequency
- **df**: document frequency
- **idf**: inverse document frequency
- **weight**: the final weight of the term in the query or document
- **n’lized**: document weights after cosine normalization
- **product**: the product of final query weight and final document weight

**Summary:** Ranked retrieval in the vector space model

- Represent the query as a weighted tf-idf vector
- Represent each document as a weighted tf-idf vector
- Compute the cosine similarity between the query vector and each document vector
- Rank documents with respect to the query
- Return the top \( K \) (e.g., \( K = 10 \)) to the user
Resources

- Chapters 6 and 7 of IIR
- Resources at http://ifnlp.org/ir
- Vector space for dummies
- Exploring the similarity space (Moffat and Zobel, 2005)
- Okapi BM25 (a state-of-the-art weighting method, 11.4.3 of IIR)