### Introduction to Information Retrieval

**CS276: Information Retrieval and Web Search**

Pandu Nayak and Prabhakar Raghavan

**Lecture 6: Scoring, Term Weighting and the Vector Space Model**

---

### Recap of lecture 5

- Collection and vocabulary statistics: Heaps’ and Zipf’s laws
- Dictionary compression for Boolean indexes
  - Dictionary string, blocks, front coding
- Postings compression: Gap encoding, prefix-unique codes
  - Variable-Byte and Gamma codes

<table>
<thead>
<tr>
<th>Collection (text, xml markup etc)</th>
<th>3,600.0 MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection (text)</td>
<td>960.0</td>
</tr>
<tr>
<td>Term-doc incidence matrix</td>
<td>40,000.0</td>
</tr>
<tr>
<td>Postings, uncompressed (32-bit words)</td>
<td>400.0</td>
</tr>
<tr>
<td>Postings, uncompressed (20 bits)</td>
<td>250.0</td>
</tr>
<tr>
<td>Postings, variable byte encoded</td>
<td>116.0</td>
</tr>
<tr>
<td>Postings, γ-encoded</td>
<td>101.0</td>
</tr>
</tbody>
</table>

---

### This lecture; IIR Sections 6.2-6.4.3

- Ranked retrieval
- Scoring documents
- Term frequency
- Collection statistics
- Weighting schemes
- Vector space scoring

---

### 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 incapable 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.

---

### 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.
  - AND gives too few; OR gives too many

---

### Ranked retrieval models

- Rather than a set of documents satisfying a query expression, in ranked retrieval, the system returns an ordering over the (top) documents in the collection for a query
- **Free text queries**: Rather than a query language of operators and expressions, the user’s query is just one or more words in a human language
- In principle, there are two separate choices here, but in practice, ranked retrieval has normally been associated with free text queries and vice versa
Feast or famine: not a problem in ranked retrieval

- When a system produces a ranked result set, large result sets are not an issue
  - Indeed, the size of the result set is not an issue
  - We just show the top \( k \) (≈ 10) results
  - We don’t overwhelm the user
- Premise: the ranking algorithm works

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”.

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 (should be)
- We will look at a number of alternatives for this.

Take 1: Jaccard coefficient

- Recall from Lecture 3: A commonly used measure of overlap of two sets \( A \) and \( B \)
  - \( \text{jaccard}(A, B) = |A \cap B| / |A \cup B| \)
  - \( \text{jaccard}(A, A) = 1 \)
  - \( \text{jaccard}(A, B) = 0 \) 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: Scoring example

- What is the query-document match score that the Jaccard coefficient computes for each of the two documents below?
- Query: ides of march
- Document 1: caesar died in march
- Document 2: the long march

Issues with Jaccard for scoring

- It doesn’t consider term frequency (how many times a term occurs in a document)
- Rare terms in a collection 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 \( |A \cap B| / \sqrt{|A \cup B|} \)
- . . . instead of \( |A \cap B| / |A \cup B| \) (Jaccard) for length normalization.
Recall (Lecture 1): Binary term-document incidence matrix

<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>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>
</tr>
<tr>
<td>Caesar</td>
<td>1</td>
<td>1</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>
</tr>
<tr>
<td>Cleopatra</td>
<td>1</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>
</tr>
<tr>
<td>worser</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

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

Bag of words model

- Vector representation doesn’t consider the ordering of words in a document
- \textit{John is quicker than Mary} and \textit{Mary is quicker than John} have the same vectors
- This is called the 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-document count matrices

<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>157</td>
<td>73</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Brutus</td>
<td>4</td>
<td>157</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>232</td>
<td>237</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>10</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>
</tr>
<tr>
<td>mercy</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>worser</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Log-frequency weighting

- The log frequency weight of term \( t \) in document \( d \) is
  \[
  w_{t,d} = \begin{cases} 
  1 + \log_{10} t_{f,t,d}, & \text{if } t_{f,t,d} > 0 \\
  0, & \text{otherwise} 
  \end{cases}
  \]
- \( 0 \to 0, 1 \to 1, 2 \to 1.3, 10 \to 2, 1000 \to 4, \) etc.
- Score for a document-query pair: sum over terms \( t \) in both \( q \) and \( d \):
  \[
  \text{score} = \sum_{t \in q \cap d} (1 + \log t_{f,t,d})
  \]
- The score is 0 if none of the query terms is present in the document.

Term frequency tf

- The term frequency \( t_{f,t,d} \) 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 1 occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Document frequency

- Rare terms are more informative than frequent terms
  - Recall stop words
  - Consider a term in the query that is rare in the collection (e.g., \textit{arachnocentric})
- A document containing this term is very likely to be relevant to the query \textit{arachnocentric}
- We want a high weight for rare terms like \textit{arachnocentric}.
Document frequency, continued

- Frequent terms are less informative than rare terms
- Consider a query term that is frequent in the collection (e.g., *high*, *increase*, *line*)
- A document containing such a 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 high positive weights for words like *high*, *increase*, and *line*
- But lower weights than for rare terms.
- We will use document frequency (df) to capture this.

idf weight

- df, is the document frequency of t: the number of documents that contain t
- df is an inverse measure of the informativeness of t
- df ≤ N
- We define the idf (inverse document frequency) of t by
  \[ \text{idf}_t = \log_{10} \left( \frac{N}{df_t} \right) \]
- We use \( \log (N/df_t) \) instead of \( \frac{N}{df_t} \) to “dampen” the effect of idf.

\[ \text{idf}_t = \log_{10} \left( \frac{N}{df_t} \right) \]

There is one idf value for each term t in a collection.

Effect of idf on ranking

- Does idf have an effect on ranking for one-term queries, like
  - iPhone
- idf has no effect on ranking one term queries
  - idf affects the ranking of documents for queries with at least two terms
  - For the query *capricious person*, idf weighting makes occurrences of *capricious* count for much more in the final document ranking than occurrences of *person*.

Collection vs. Document frequency

- The collection frequency of t is the number of occurrences of t in the collection, counting multiple occurrences.
- Example:

<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>

- Which word is a better search term (and should get a higher weight)?

<table>
<thead>
<tr>
<th>Word</th>
<th>tf-idf weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>insurance</td>
<td>( \text{tf-idf} ) + ( \text{idf} )</td>
</tr>
<tr>
<td>try</td>
<td>( \text{tf-idf} ) × ( \text{idf} )</td>
</tr>
</tbody>
</table>

Increasing with the number of occurrences within a document
Increasing with the rarity of the term in the collection
Score for a document given a query

\[ \text{Score}(q, d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d} \]

- There are many variants
  - How “tf” is computed (with/without logs)
  - Whether the terms in the query are also weighted
  - ...

Documents as vectors

- So we have a $|V|$-dimensional 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
- These are very sparse vectors - 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 in this space
- **proximity = similarity of vectors**
- **proximity \( \approx \) inverse of distance**
- **Recall:** We do 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

Why distance is a bad idea

The Euclidean distance between $q$ and $d$ is large even though the distribution of terms in the query $q$ and the distribution of terms in the document $d$ are very similar.
Use angle instead of distance

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

- Key idea: Rank documents according to angle with query.

From angles to cosines

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

Length normalization

- A vector can be (length-) normalized by dividing each of its components by its length – for this we use the $L_2$ norm: 
  $$\|\mathbf{v}\|_2 = \sqrt{\sum_{i=1}^{\|\mathbf{v}\|} v_i^2}$$
- Dividing a vector by its $L_2$ norm makes it a unit (length) vector (on surface of unit hypersphere)
- Effect on the two documents $d$ and $d'$ ($d$ appended to itself) from earlier slide: they have identical vectors after length-normalization.
  - Long and short documents now have comparable weights

Cosine for length-normalized vectors

- For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):
  $$\cos(\mathbf{\hat{q}}, \mathbf{\hat{d}}) = \mathbf{\hat{q}} \cdot \mathbf{\hat{d}} = \sum_{i=1}^{\|\mathbf{q}\|} q_i d_i$$
  for $\mathbf{\hat{q}}, \mathbf{\hat{d}}$ length-normalized.
Cosine similarity illustrated

![Diagram of cosine similarity between two vectors](image)

**POOR**

\[ \vec{v}(d_1) \]

\[ \vec{v}(q) \]

\[ \vec{v}(d_2) \]

\[ \vec{v}(d_3) \]

**RICH**

How similar are the novels

**SaS:** Sense and Sensibility

**PaP:** Pride and Prejudice, and

**WH:** Wuthering Heights?

Term frequencies (counts)

<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>

Note: To simplify this example, we don’t do idf weighting.

Computing cosine scores

**cosineScore(q)**

1. `float Scores[N] = 0`
2. `float Length[N]`
3. `for each query term t`
4. `do calculate w_t,q and fetch postings list for t`
5. `for each pair(d, t) in postings list`
6. `do Scores[d] += w_t,d \times w_t,q`
7. `Read the array Length`
8. `for each d`
10. `return Top K components of Scores[]`

Why do we have `cos(SaS,PaP) > cos(SaS,WH)`?

\[
\cos(SaS,PaP) \approx 0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0 \\
\approx 0.94
\]

\[
\cos(SaS,WH) \approx 0.79
\]

\[
\cos(PaP,WH) \approx 0.69
\]

**tf-idf weighting has many variants**

<table>
<thead>
<tr>
<th>Term frequency</th>
<th>Document frequency</th>
<th>Normalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t(f_i) )</td>
<td>( \theta(d_j) )</td>
<td>( t(f_i) )</td>
</tr>
<tr>
<td>( s ) (augmented) ( 0.5 + \frac{\log tf_i + 1}{1 + \log d_j + 1} ) ( s ) (prob idf) ( \max(0, \frac{\log \frac{1.5 + \log tf_i + 1}{1 + \log d_j + 1}}{\log d_j}) ) ( c ) (cosine) ( \frac{\sqrt{\sum \sqrt{t(f_i) \cdot \theta(d_j)}}}{\sqrt{\sum \sqrt{t(f_i) \cdot \theta(d_j)}}} )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( b ) (boost) ( \begin{cases} 1 &amp; \text{if} t(f_i) &gt; 0 \ 0 &amp; \text{otherwise} \end{cases} ) ( b ) (size) ( \frac{1}{\text{size}_\theta(d_j)} ) ( \alpha &lt; 1 )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Columns headed ‘n’ are acronyms for weight schemes.

Why is the base of the log in idf immaterial?
tf-idf example: Inc.ltc

Document: car insurance auto insurance
Query: best car insurance

<table>
<thead>
<tr>
<th>Term</th>
<th>Query: best car insurance</th>
<th>Document: car insurance auto insurance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tf-raw</td>
<td>idf</td>
</tr>
<tr>
<td>auto</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>best</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>car</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>insurance</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Exercise: what is $N$, the number of docs?

Doc length = $\sqrt{2.3^2 + 0^2 + 1^2 + 1.3^2} = 1.92$

Score = $0 + 0 + 0.27 + 0.53 = 0.8$

Summary – vector space ranking

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

Resources for today’s lecture

- IIR 6.2 – 6.4.3
  - Term weighting and cosine similarity tutorial for SEO folk!