COMS E6111  
Advanced Database Systems  
Fall 2017  

Computer Science Department  
Columbia University  

Acknowledgement  

Slides borrowed and adapted from the authors of:  

Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze: *Introduction to Information Retrieval*, Cambridge University Press, 2008  

and from Stanford’s CS276: Information Retrieval and Web Search course, by Pandu Nayak and Prabhakar Raghavan  

Original slides available electronically from  

Information Retrieval

Goal: Search over a “database” of **text** documents
- web pages
- library catalog entries
- research papers
- news articles
- email, files, ...
- ...

Query? In simplest incarnation, a list of words:
[**data mining**]

Information Retrieval

Problem: Mapping from a user’s “information need” into the items in the text database that satisfy the need

... or/and, most recently, into **information extracted or synthesized** from the database contents
Information Retrieval Models (many!)

- Boolean (“set theoretic”)
- Vector-space (“algebraic”)
- Probabilistic
- Language Modeling
- ...

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: [a AND b] → 200,000 hits
- Query 2: [a AND b AND c]: 0 hits
- Challenging 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
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 $k$ ($\approx 10$) 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

- A commonly used measure of overlap of two sets A and B
- \( \text{jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|} \)
- \( \text{jaccard}(A, A) = 1 \)
- \( \text{jaccard}(A, B) = 0 \) if \( A \cap B = \text{empty set} \)
- \( A \) 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
Consider a binary term-document “incidence matrix”

<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>

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

---

Term-document count matrices

Now consider the number of occurrences of a term in a document:
- Each document is a count vector in \( \mathbb{N}^{|V|} \): a column below

<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>157</td>
<td>73</td>
<td>0</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>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>232</td>
<td>227</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>10</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>
</tr>
<tr>
<td>mercy</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>worser</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
**Bag of words model**

- Vector representation doesn’t consider the ordering of words in a document
- *John is quicker than Mary* and *Mary is quicker than John* have the same vectors
- This is called the bag of words model.
- We will look at capturing positional information later.
- For now: bag of words model

---

**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.
Log-frequency weighting

- The log frequency weight of term $t$ in $d$ is
  \[
  w_{t,d} = \begin{cases} 
  1 + \log_{10} tf_{t,d} , & \text{if } tf_{t,d} > 0 \\
  0 , & \text{otherwise}
  \end{cases}
  \]
- $0 \to 0$, $1 \to 1$, $2 \to 1.3$, $10 \to 2$, $1000 \to 4$, etc.
- Score for a document-query pair: sum over terms $t$ in both $q$ and $d$:
  \[
  \text{score} = \sum_{t \in q \cap d} (1 + \log tf_{t,d})
  \]
- 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
  - Recall stop words
- 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 to the query *arachnocentric*
- $\to$ We want a high weight for rare terms like *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.

Sec. 6.2.1

idf weight

- $df_t$ is the document frequency of $t$: the number of documents that contain $t$
  - $df_t$ is an inverse measure of the informativeness of $t$
  - $df_t \leq N$, where $N$ is the number of documents in the collection
- We define the idf (inverse document frequency) of $t$ by
  $$idf_t = \log_{10} \left( \frac{N}{df_t} \right)$$
  - We use $\log (N/df_t)$ instead of $N/df_t$ to “dampen” the effect of idf.
### 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)?

tf-idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.

$$ w_{t,d} = \log(1 + tf_{t,d}) \times \log_{10}(N / df_t) $$

- Best known weighting scheme in information retrieval
  - Note: the “-” in tf-idf is a hyphen, not a minus sign!
  - Alternative names: tf.idf, tf x idf

- Increases with the number of occurrences within a document
- Increases 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
  - ...

Sec. 6.2.2

Binary \rightarrow \text{count} \rightarrow \text{weight matrix}

<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>5.25</td>
<td>3.18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.35</td>
</tr>
<tr>
<td>Brutus</td>
<td>1.21</td>
<td>6.1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>8.59</td>
<td>2.54</td>
<td>0</td>
<td>1.51</td>
<td>0.25</td>
<td>0</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1.54</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>2.85</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>1.51</td>
<td>0</td>
<td>1.9</td>
<td>0.12</td>
<td>5.25</td>
<td>0.88</td>
</tr>
<tr>
<td>worser</td>
<td>1.37</td>
<td>0</td>
<td>0.11</td>
<td>4.15</td>
<td>0.25</td>
<td>1.95</td>
</tr>
</tbody>
</table>

Each document is now represented by a real-valued vector of tf-idf weights \( \in \mathbb{R}^{|V|} \)
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
Formalizing vector space proximity

- 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 Euclidean distance is a bad idea

The Euclidean distance between $\vec{q}$ and $\vec{d}_2$ is large even though the distribution of terms in the query $\vec{q}$ and the distribution of terms in the document $\vec{d}_2$ 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 $[0^\circ, 180^\circ]$
From angles to cosines

But how – and why – should we be computing cosines?

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:
  \[
  \| \mathbf{x} \|_2 = \sqrt{\sum_i x_i^2}
  \]

- 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(query, document)

Dot product

Unit vectors

\[
\cos(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{|\vec{q}| |\vec{d}|} = \frac{q_i d_i}{\sqrt{\sum_{i=1}^{|
u|} q_i^2} \sqrt{\sum_{i=1}^{|
u|} d_i^2}}
\]

- \(q_i\) is the tf-idf weight of term \(i\) in the query
- \(d_i\) is the tf-idf weight of term \(i\) in the document

\(\cos(\vec{q}, \vec{d})\) is the cosine similarity of \(\vec{q}\) and \(\vec{d}\) ... or, equivalently, the cosine of the angle between \(\vec{q}\) and \(\vec{d}\).

Cosine for length-normalized vectors

- For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):

\[
\cos(\vec{q}, \vec{d}) = \vec{q} \cdot \vec{d} = \sum_{i=1}^{|
u|} q_i d_i
\]

for \(q, d\) length-normalized.
Problem with document length normalization?

Pivoted Document Length Normalization

Goal: Avoid “hurting” long documents unfairly
### 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>n (natural)</td>
<td>n (no)</td>
<td>n (none)</td>
</tr>
<tr>
<td>l (logarithm)</td>
<td>1 + \log(t_{fr,d})</td>
<td>t (idf)</td>
</tr>
<tr>
<td>a (augmented)</td>
<td>0.5 \times \frac{t_{fr,d}}{\max(t_{fr,d})}</td>
<td>p (prob idf)</td>
</tr>
<tr>
<td>b (boolean)</td>
<td>\begin{cases} 1 &amp; \text{if } t_{fr,d} &gt; 0 \ 0 &amp; \text{otherwise} \end{cases}</td>
<td>u (pivoted unique)</td>
</tr>
<tr>
<td>L (log ave)</td>
<td>\frac{1 + \log(t_{fr,d})}{\log(\text{avg}(t_{fr,d}))}</td>
<td>b (byte size)</td>
</tr>
</tbody>
</table>

| t (idf) | \log \frac{N}{df_t} |
| p (prob idf) | \max(0, \log \frac{N_{id}}{df_t}) |
| u (pivoted unique) | 1/\sqrt{w_1^2 + w_2^2 + \ldots + w_T^2} |
| b (byte size) | 1/\text{CharLength}^\alpha, \alpha < 1 |

### Weighting may differ in queries vs documents

- Many search engines allow for different weightings for queries vs. documents
- SMART Notation: denotes the combination in use in an engine, with the notation \textit{ddd.qqq}, using the acronyms from the previous table
- A very standard weighting scheme is: Inc.ltc
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

Problem: Queries are often ambiguous

- [jaguar]?
- [tiger]?
- [jobs]?
- [gravano]?
- ...
“Relevance feedback”

1. User issues a (potentially ambiguous) query
2. Then user marks each document among the top results as relevant or not relevant to the query
3. Retrieval system uses this “relevance feedback” from the user to refine query and produce new, hopefully improved query results
4. If necessary, steps 2 and 3 are repeated.

Rocchio’s algorithm
Additional factors for ranking: Discussion

Alternative to vector-space retrieval model: the probabilistic model

Given a user query q and a document d, estimate probability(d is relevant for q)
Goal: maximize estimate over the set of relevant documents R, and minimize it outside of R