Introduction to Information Retrieval
IIR 6: Scoring, Term Weighting, The Vector Space Model

Paul Gray

Department of Computer Science

2017-03-02
Overview

1. Recap
2. Why ranked retrieval?
3. Term frequency
4. tf-idf weighting
5. The vector space model
Why ranked retrieval? Term frequency tf-idf weighting The vector space model

Heaps’ law

Vocabulary size $M$ as a function of collection size $T$ (number of tokens) for Reuters-RCV1. For these data, the dashed line

$$\log_{10} M = 0.49 \times \log_{10} T + 1.64$$

is the best least squares fit.

Thus, $M = 10^{1.64} T^{0.49}$ and $k = 10^{1.64} \approx 44$ and $b = 0.49$. 

log10 M

0 1 2 3 4 5 6

log10 T

0 2 4 6 8
Zipf’s law

$$cf_i \propto \frac{1}{i}$$

The most frequent term (*the*) occurs $cf_1$ times, the second most frequent term (*of*) occurs $cf_2 = \frac{1}{2}cf_1$ times, the third most frequent term (*and*) occurs $cf_3 = \frac{1}{3}cf_1$ times etc.
Dictionary as a string

...stil.sl.sygetic.syzygial.syzygyszai.beryiteszecinszono...

freq. postings ptr. term ptr.
9 →
92 →
5 →
71 →
12 →
...
...
...

4 bytes 4 bytes 3 bytes
# Gap encoding

<table>
<thead>
<tr>
<th>term</th>
<th>encoding</th>
<th>postings list</th>
</tr>
</thead>
<tbody>
<tr>
<td>THE</td>
<td>docIDs</td>
<td>283042</td>
</tr>
<tr>
<td></td>
<td>gaps</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>283043</td>
</tr>
<tr>
<td></td>
<td></td>
<td>283044</td>
</tr>
<tr>
<td></td>
<td></td>
<td>283045</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>COMPUTER</td>
<td>docIDs</td>
<td>283047</td>
</tr>
<tr>
<td></td>
<td>gaps</td>
<td>107</td>
</tr>
<tr>
<td></td>
<td></td>
<td>283154</td>
</tr>
<tr>
<td></td>
<td></td>
<td>283159</td>
</tr>
<tr>
<td></td>
<td></td>
<td>283202</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>ARACHNOCENTRIC</td>
<td>docIDs</td>
<td>252000</td>
</tr>
<tr>
<td></td>
<td>gaps</td>
<td>252000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>248100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>500100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
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

- Represent a gap $G$ as a pair of **length** and **offset**.
- Offset is the gap in binary, with the leading bit chopped off.
- Length is the length of offset.
- Encode length in **unary** code
- The Gamma code is the concatenation of length and offset.
## Compression of Reuters

<table>
<thead>
<tr>
<th>data structure</th>
<th>size in MB</th>
</tr>
</thead>
<tbody>
<tr>
<td>dictionary, fixed-width</td>
<td>11.2</td>
</tr>
<tr>
<td>dictionary, term pointers into string</td>
<td>7.6</td>
</tr>
<tr>
<td>~, with blocking, $k = 4$</td>
<td>7.1</td>
</tr>
<tr>
<td>~, with blocking &amp; front coding</td>
<td>5.9</td>
</tr>
<tr>
<td>collection (text, xml markup etc)</td>
<td>3600.0</td>
</tr>
<tr>
<td>collection (text)</td>
<td>960.0</td>
</tr>
<tr>
<td>T/D 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, $\gamma$ encoded</td>
<td>101.0</td>
</tr>
</tbody>
</table>
Take-away today
Take-away today

- **Ranking** search results: why it is important (as opposed to just presenting a set of unordered Boolean results)
Take-away today

- **Ranking** search results: why it is important (as opposed to just presenting a set of unordered Boolean results)
- **Term frequency**: This is a key ingredient for ranking.
Take-away today

- **Ranking** search results: why it is important (as opposed to just presenting a set of unordered Boolean results)
- **Term frequency**: This is a key ingredient for ranking.
- **Tf-idf ranking**: best known traditional ranking scheme
Take-away today

- **Ranking** search results: why it is important (as opposed to just presenting a set of unordered Boolean results)
- **Term frequency**: This is a key ingredient for ranking.
- **Tf-idf ranking**: best known traditional ranking scheme
- **Vector space model**: One of the most important formal models for information retrieval (along with Boolean and probabilistic models)
Outline

1 Recap

2 Why ranked retrieval?

3 Term frequency

4 tf-idf weighting

5 The vector space model
Ranked retrieval
ranked retrieval

Thus far, our queries have been **Boolean**.
Documents either match or don’t.

**Good for expert users** with precise understanding of their needs and of 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.
Problem with Boolean search: Feast or famine
Boolean queries often result in either too few (0) or too many (1000s) results.

Query 1 (boolean conjunction): [standard user dlink 650]
  → 200,000 hits – feast

Query 2 (boolean conjunction): [standard user dlink 650 no card found]
  → 0 hits – famine

In Boolean retrieval, it takes a lot of skill to come up with a query that produces a manageable number of hits.
Feast or famine: No problem in ranked retrieval
Feast or famine: No problem in ranked retrieval

- With ranking, large result sets are not an issue.
- Just show the top 10 results
- Doesn’t overwhelm the user
- Premise: the ranking algorithm works: More relevant results are ranked higher than less relevant results.
### Scoring as the basis of ranked retrieval

<table>
<thead>
<tr>
<th>Recap</th>
<th>Why ranked retrieval?</th>
<th>Term frequency</th>
<th>tf-idf weighting</th>
<th>The vector space model</th>
</tr>
</thead>
</table>

Gray: Scoring, term weighting, the vector space model
Scoring as the basis of ranked retrieval

- We wish to rank documents that are more relevant higher than documents that are less relevant.
- How can we accomplish such a ranking of the documents in the collection with respect to a query?
- Assign a score to each query-document pair, say in $[0, 1]$.
- This score measures how well document and query “match”.
Query-document matching scores

- How do we compute the score of 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
Take 1: Jaccard coefficient

- A commonly used measure of overlap of two sets
- Let $A$ and $B$ be two sets
- Jaccard coefficient:

$$\text{JACCARD}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

$(A \neq \emptyset \text{ or } B \neq \emptyset)$

- $\text{JACCARD}(A, A) = 1$
- $\text{JACCARD}(A, B) = 0$ 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
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”
  - \( \text{JACCARD}(q, d) = \frac{1}{6} \)
What’s wrong with Jaccard?
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 does not consider this information.
- We need a more sophisticated way of normalizing for the length of a document.
- Later in this lecture, we’ll use \( \frac{|A \cap B|}{\sqrt{|A \cup B|}} \) (cosine) . . .
- . . . instead of \( \frac{|A \cap B|}{|A \cup B|} \) (Jaccard) for length normalization.
Outline

1 Recap

2 Why ranked retrieval?

3 Term frequency

4 tf-idf weighting

5 The vector space model
### Binary incidence 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><strong>Anthony</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>Brutus</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>Caesar</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>Calpurnia</strong></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>Cleopatra</strong></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>Mercy</strong></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>Worse</strong></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Each document is represented as a binary vector \( \in \{0, 1\}^{|V|} \).
### Binary incidence 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>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ANTHONY</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>BRUTUS</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>CAESAR</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>CALPURNIA</strong></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>CLEOPATRA</strong></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>MERCY</strong></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>WORSER</strong></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 as a *binary vector* $\in \{0, 1\}^{|V|}$. 
## Count matrix

<table>
<thead>
<tr>
<th></th>
<th>Anthony and Caesar</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><strong>Anthony</strong></td>
<td>157</td>
<td>73</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>Brutus</strong></td>
<td>4</td>
<td>157</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>Caesar</strong></td>
<td>232</td>
<td>227</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>Calpurnia</strong></td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>Cleopatra</strong></td>
<td>57</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>Mercy</strong></td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>8</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td><strong>Worse</strong></td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

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

---

**Recap**

Why ranked retrieval?

**Term frequency**

- **tf-idf weighting**
- **The vector space model**
### Count 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>157</td>
<td>73</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></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></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></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></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></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></td>
</tr>
<tr>
<td>Worse</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Each document is now represented as a count vector \( \in \mathbb{N}^{|V|} \).
Bag of words model
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 because:
  - A document with \( tf = 10 \) occurrences of the term is more relevant than a document with \( tf = 1 \) occurrence of the term.
  - But not 10 times more relevant.
  - Relevance does not increase proportionally with term frequency.
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 because:

A document with $tf = 10$ occurrences of the term is more relevant than a document with $tf = 1$ occurrence of the term.

But not 10 times more relevant.

Relevance does not increase proportionally with term frequency.
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 because:
  - A document with \( tf = 10 \) occurrences of the term is more relevant than a document with \( tf = 1 \) occurrence of the term.
  - But not 10 times more relevant.
  - Relevance does not increase proportionally with term frequency.
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 because:
  - A document with $tf = 10$ occurrences of the term is more relevant than a document with $tf = 1$ occurrence of the term.
  - But not 10 times more relevant.
  - Relevance does not increase proportionally with term frequency.
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 because:
  - A document with \( tf = 10 \) occurrences of the term is more relevant than a document with \( tf = 1 \) occurrence of the term.
  - But not 10 times more relevant.
  - Relevance does not increase proportionally with term frequency.
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 because:

A document with $tf = 10$ occurrences of the term is more relevant than a document with $tf = 1$ occurrence of the term.

But not 10 times more relevant.

Relevance does not increase proportionally with term frequency.
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 because:

A document with $tf = 10$ occurrences of the term is more relevant than a document with $tf = 1$ occurrence of the term.

But not 10 times more relevant.

Relevance does not increase proportionally with term frequency.
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 because:

A document with $tf = 10$ occurrences of the term is more relevant than a document with $tf = 1$ occurrence of the term.

But not 10 times more relevant.

Relevance does not increase proportionally with term frequency.
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 $t_f$ when computing query-document match scores.

But how?

Raw term frequency is not what we want because:

A document with $t_f = 10$ occurrences of the term is more relevant than a document with $t_f = 1$ occurrence of the term.

But not 10 times more relevant.

Relevance does not increase proportionally with term frequency.
Instead of raw frequency: Log frequency weighting
Instead of raw frequency: Log frequency weighting

- The log frequency weight of term \( t \) in \( d \) is defined as follows:

\[
w_{t,d} = \begin{cases} 
1 + \log_{10} tf_{t,d} & \text{if } tf_{t,d} > 0 \\
0 & \text{otherwise}
\end{cases}
\]

- \( tf_{t,d} \rightarrow w_{t,d} \):
  - \( 0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4 \), etc.

- Score for a document-query pair: sum over terms \( t \) in both \( q \) and \( d \):
  \[
  \text{tf-matching-score}(q, d) = \sum_{t \in q \cap d} (1 + \log \text{tf}_{t,d})
  \]

- The score is 0 if none of the query terms is present in the document.
Exercise

- Compute the Jaccard matching score and the tf matching score for the following query-document pairs.
  - q: [information on cars] d: “all you’ve ever wanted to know about cars”
  - q: [information on cars] d: “information on trucks, information on planes, information on trains”
  - q: [red cars and red trucks] d: “cops stop red cars more often”
Outline

1 Recap
2 Why ranked retrieval?
3 Term frequency
4 tf-idf weighting
5 The vector space model
Recap Why ranked retrieval? Term frequency tf-idf weighting The vector space model

Frequency in document vs. frequency in collection
In addition, to term frequency (the frequency of the term in the document) . . .

. . . we also want to use the frequency of the term in the collection for weighting and ranking.
Desired weight for rare terms
Desired weight for rare terms

- 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 high weights for rare terms like ARACHNOCENTRIC.
Desired weight for frequent terms
Desired weight for frequent terms

- Frequent terms are less informative than rare terms.
- Consider a term in the query that is frequent in the collection (e.g., GOOD, INCREASE, LINE).
- A document containing this term is more likely to be relevant than a document that doesn’t . . .
- . . . but words like GOOD, INCREASE and LINE are not sure indicators of relevance.
- → For frequent terms like GOOD, INCREASE, and LINE, we want positive weights . . .
- . . . but lower weights than for rare terms.
Document frequency

- We want high weights for rare terms like ARACHNOCENTRIC.
- We want low (positive) weights for frequent words like GOOD, INCREASE, and LINE.
We want high weights for rare terms like ARACHNOCENTRIC.

We want low (positive) weights for frequent words like GOOD, INCREASE, and LINE.

We will use document frequency to factor this into computing the matching score.
We want high weights for rare terms like ARACHNOCENTRIC.

We want low (positive) weights for frequent words like GOOD, INCREASE, and LINE.

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.
Recap Why ranked retrieval? Term frequency **tf-idf weighting** The vector space model

idf weight
idf weight

- \( df_t \) is the document frequency, the number of documents that \( t \) occurs in.
- \( df_t \) is an inverse measure of the informativeness of term \( t \).
- We define the idf weight of term \( t \) as follows:

\[
idf_t = \log_{10} \frac{N}{df_t}
\]

\( N \) is the number of documents in the collection.

- \( idf_t \) is a measure of the informativeness of the term.
- \([\log N/df_t]\) instead of \([N/df_t]\) to “dampen” the effect of idf
- Note that we use the log transformation for both term frequency and document frequency.
Examples for idf
Examples for idf

Compute $idf_t$ using the formula: 

$$idf_t = \log_{10} \frac{1,000,000}{df_t}$$

<table>
<thead>
<tr>
<th>term</th>
<th>$df_t$</th>
<th>$idf_t$</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
Effect of idf on ranking

- idf affects the ranking of documents for queries with at least two terms.
- For example, in the query “arachnocentric line”, idf weighting increases the relative weight of ARACHNOCENTRIC and decreases the relative weight of LINE.
- idf has little effect on ranking for one-term queries.
## 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>

- **Collection frequency of** $t$: number of tokens of $t$ in the collection
- **Document frequency of** $t$: number of documents $t$ occurs in
# 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>

- Collection frequency of $t$: number of tokens of $t$ in the collection
- Document frequency of $t$: number of documents $t$ occurs in
- Why these numbers?
Recap Why ranked retrieval? Term frequency tf-idf weighting The vector space model

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>

- Collection frequency of $t$: number of tokens of $t$ in the collection
- Document frequency of $t$: number of documents $t$ occurs in
- Why these numbers?
- Which word is a better search term (and should get a higher weight)?
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>

- Collection frequency of $t$: number of tokens of $t$ in the collection
- Document frequency of $t$: number of documents $t$ occurs in
- Why these numbers?
- Which word is a better search term (and should get a higher weight)?
- This example suggests that df (and idf) is better for weighting than cf (and “icf”).
Recap Why ranked retrieval? Term frequency tf-idf weighting The vector space model

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

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

tf-weight idf-weight

- Best known weighting scheme in information retrieval
- Note: the “-” in tf-idf is a hyphen, not a minus sign!
- Alternative names: tf.idf, tf \times idf
The tf-idf weight of a term is the product of its tf weight and its idf weight.

\[ w_{t,d} = (1 + \log tf_{t,d}) \cdot \log \frac{N}{df_t} \]

tf-weight idf-weight

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

\[
wt,d = (1 + \log tf_{t,d}) \cdot \log \frac{N}{df_t}
\]

tf-weight idf-weight

- 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
tf-idf weighting

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

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

- tf-weight idf-weight
- 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
Recap Why ranked retrieval? Term frequency tf-idf weighting The vector space model

Summary: tf-idf
Assign a tf-idf weight for each term $t$ in each document $d$:

$$w_{t,d} = (1 + \log tf_{t,d}) \cdot \log \frac{N}{df_t}$$

The tf-idf weight . . .

- . . . increases with the number of occurrences within a document. (term frequency)
- . . . increases with the rarity of the term in the collection. (inverse document frequency)
Exercise: 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 $cf$?
- Relationship between $tf$ and $df$?
### Binary incidence matrix

<table>
<thead>
<tr>
<th></th>
<th>Anthony and Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Anthony</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td><strong>Brutus</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Caesar</strong></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Calpurnia</strong></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Cleopatra</strong></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Mercy</strong></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>Worsomer</strong></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 as a binary vector $\in \{0, 1\}^{|V|}$. 
<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>157</td>
<td>73</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></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></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></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></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></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></td>
</tr>
<tr>
<td>WORSEMER</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$. 
### Binary → count → weight matrix

<table>
<thead>
<tr>
<th>Character</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><strong>ANTHONY</strong></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><strong>BRUTUS</strong></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><strong>CAESAR</strong></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><strong>CALPURNIA</strong></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><strong>CLEOPATRA</strong></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><strong>MERCY</strong></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><strong>WORSER</strong></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 as a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$. 
### 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>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ANTHONY</strong></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>
</tr>
<tr>
<td><strong>BRUTUS</strong></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>
</tr>
<tr>
<td><strong>CAESAR</strong></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>
</tr>
<tr>
<td><strong>CALPURNIA</strong></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>
</tr>
<tr>
<td><strong>CLEOPATRA</strong></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>
</tr>
<tr>
<td><strong>MERCY</strong></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>
</tr>
<tr>
<td><strong>WORSER</strong></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>
</tr>
</tbody>
</table>

Each document is now represented as a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$. 

---

Gray: Scoring, term weighting, the vector space model
Documents as vectors
Documents as vectors

- Each document is now represented as 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 web search engines.
- Each vector is very sparse - most entries are zero.
Queries as vectors
Queries as vectors

- Key idea 1: do the same for queries: represent them as vectors in the high-dimensional space
- Key idea 2: Rank documents according to their proximity to the query
  - proximity = similarity
  - proximity ≈ negative distance
- Recall: We’re doing this because we want to get away from the you’re-either-in-or-out, feast-or-famine Boolean model.
- Instead: rank relevant documents higher than nonrelevant documents
How do we formalize vector space similarity?
How do we formalize vector space similarity?

- First cut: (negative) distance between two points
  
  \(( = \text{distance between the end points of the two vectors})\)

- Euclidean distance?

- Euclidean distance is a bad idea . . .

- . . . because Euclidean distance is large for vectors of different lengths.
Why distance is a bad idea
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
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 $d'$. $d'$ is twice as long as $d$.
- “Semantically” $d$ and $d'$ have the same content.
- The angle between the two documents is 0, corresponding to maximal similarity . . .
- . . . even though 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 $\text{cosine}(\text{query}, \text{document})$ in increasing order

Cosine is a monotonically decreasing function of the angle for the interval $[0^\circ, 180^\circ]$
Cosine
Recap Why ranked retrieval? Term frequency tf-idf weighting

The vector space model

Cosine

Graph showing a cosine function.
Length normalization
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 similarity between query and document
Cosine similarity between query and document

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

- \(q_i\) 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 \(\vec{d}\).
For normalized vectors, the cosine is equivalent to the dot product or scalar product.

\[ \cos(\vec{q}, \vec{d}) = \vec{q} \cdot \vec{d} = \sum_i q_i \cdot d_i \]

(if \( \vec{q} \) and \( \vec{d} \) are length-normalized).
Cosine similarity illustrated
Cosine similarity illustrated

\[ \vec{v}(d_1), \vec{v}(d_2), \vec{v}(d_3), \vec{v}(q) \]

\[ \theta \]

POOR

RICH
How similar are these novels?

SaS: Sense and Sensibility
PaP: Pride and Prejudice
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>
Cosine: Example

<table>
<thead>
<tr>
<th>term frequencies (counts)</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>term</td>
<td>SaS</td>
<td>PaP</td>
<td>WH</td>
</tr>
<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>

| log frequency weighting |
|--------------------------|---|---|---|
| term                    | SaS | PaP | WH |
| AFFECTION               | 3.06 | 2.76 | 2.30 |
| JEALOUS                 | 2.0 | 1.85 | 2.04 |
| GOSSIP                  | 1.30 | 0 | 1.78 |
| WUTHERING               | 0 | 0 | 2.58 |

(To simplify this example, we don’t do idf weighting.)
**Cosine: Example**

<table>
<thead>
<tr>
<th>term</th>
<th>SaS</th>
<th>PaP</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFFECTION</td>
<td>3.06</td>
<td>2.76</td>
<td>2.30</td>
</tr>
<tr>
<td>JEALOUS</td>
<td>2.0</td>
<td>1.85</td>
<td>2.04</td>
</tr>
<tr>
<td>GOSSIP</td>
<td>1.30</td>
<td>0</td>
<td>1.78</td>
</tr>
<tr>
<td>WUTHERING</td>
<td>0</td>
<td>0</td>
<td>2.58</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>term</th>
<th>SaS</th>
<th>PaP</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFFECTION</td>
<td>0.789</td>
<td>0.832</td>
<td>0.524</td>
</tr>
<tr>
<td>JEALOUS</td>
<td>0.515</td>
<td>0.555</td>
<td>0.465</td>
</tr>
<tr>
<td>GOSSIP</td>
<td>0.335</td>
<td>0.0</td>
<td>0.405</td>
</tr>
<tr>
<td>WUTHERING</td>
<td>0.0</td>
<td>0.0</td>
<td>0.588</td>
</tr>
</tbody>
</table>

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

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

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

**Why do we have** \( \cos(SaS, PaP) > \cos(SaS, WH) \)?
Computing the cosine score
Computing the cosine score

**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. do for each pair \((d, tf_{t,d})\) 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[]
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)</td>
<td>n (no)</td>
<td>n (none)</td>
</tr>
<tr>
<td>t (logarithm)</td>
<td>t (idf)</td>
<td>c (cosine)</td>
</tr>
<tr>
<td>a (augmented)</td>
<td>p (prob idf)</td>
<td>u (pivoted unique)</td>
</tr>
<tr>
<td>b (boolean)</td>
<td></td>
<td>b (byte size)</td>
</tr>
<tr>
<td>L (log ave)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Best known combination of weighting options

Default: no weighting
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) $t_f,d$</td>
<td>n (no) 1</td>
<td>n (none) 1</td>
</tr>
<tr>
<td>l (logarithm) $1 + \log(t_f,d)$</td>
<td>t (idf) $\log \frac{N}{df_t}$</td>
<td>c (cosine) $\frac{1}{\sqrt{w_1^2 + w_2^2 + ... + w_M^2}}$</td>
</tr>
<tr>
<td>a (augmented) $0.5 + \frac{0.5 \times t_f,d}{\max_(t_f,d)}$</td>
<td>p (prob idf) $\max{0, \log \frac{N-df_t}{df_t}}$</td>
<td>u (pivoted unique) $1/u$</td>
</tr>
<tr>
<td>b (boolean) [ \begin{cases} 1 &amp; \text{if } t_f,d &gt; 0 \ 0 &amp; \text{otherwise} \end{cases} ]</td>
<td></td>
<td>b (byte size) $1/\text{CharLength}^\alpha$, $\alpha &lt; 1$</td>
</tr>
<tr>
<td>L (log ave) $\frac{1+\log(t_f,d)}{1+\log(\text{ave}_{t\in d}(t_f,d))}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Best known combination of weighting options

Default: no weighting
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) $t_{f,t,d}$</td>
<td>n (no) 1</td>
<td>n (none) 1</td>
</tr>
<tr>
<td>l (logarithm) $1 + \log(t_{f,t,d})$</td>
<td>t (idf) $\log \frac{N}{df_t}$</td>
<td>c (cosine) $\frac{1}{\sqrt{w_1^2 + w_2^2 + \ldots + w_M^2}}$</td>
</tr>
<tr>
<td>a (augmented) $0.5 + \frac{0.5 \times t_{f,t,d}}{\max_t(t_{f,t,d})}$</td>
<td>p (prob idf) $\max{0, \log \frac{N}{df_t} - df_t}$</td>
<td>u (pivoted unique) $1/u$</td>
</tr>
<tr>
<td>b (boolean) $\begin{cases} 1 &amp; \text{if } t_{f,t,d} &gt; 0 \ 0 &amp; \text{otherwise} \end{cases}$</td>
<td></td>
<td>b (byte size) $1/\text{CharLength}^\alpha$, $\alpha &lt; 1$</td>
</tr>
<tr>
<td>L (log ave) $\frac{1 + \log(t_{f,t,d})}{1 + \log(\text{aver}<em>d(t</em>{f,t,d}))}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Best known combination of weighting options

Default: no weighting
<table>
<thead>
<tr>
<th>Recap</th>
<th>Why ranked retrieval?</th>
<th>Term frequency</th>
<th>tf-idf weighting</th>
<th>The vector space model</th>
</tr>
</thead>
</table>

**tf-idf example**
tf-idf example

- We often use **different weightings** for queries and documents.
- Notation: ddd.qqq
- Example: Inc.ltn
- document: logarithmic tf, no df weighting, cosine normalization
- query: logarithmic tf, idf, no normalization
- Isn’t it bad to not normalize the query?
tf-idf example: inc.ltn

Query: “best car insurance”. Document: “car insurance auto insurance”.

<table>
<thead>
<tr>
<th>word</th>
<th>query</th>
<th>document</th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tf-raw</td>
<td>tf-wght</td>
<td>df</td>
</tr>
<tr>
<td>auto</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>best</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>1.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>insurance</td>
<td>2.04</td>
<td></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

\[
\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92
\]
\[
\frac{1}{1.92} \approx 0.52
\]
\[
\frac{1.3}{1.92} \approx 0.68
\]

Final similarity score between query and document: \( \sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08 \)

Questions?
### tf-idf example: inc.ltn

Query: “best car insurance”. Document: “car insurance auto insurance”.

<table>
<thead>
<tr>
<th>word</th>
<th>query</th>
<th></th>
<th>document</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tf-raw</td>
<td>tf-wght</td>
<td>df</td>
<td>idf</td>
</tr>
<tr>
<td>auto</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>best</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>1.04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insurance</td>
<td>2.04</td>
<td></td>
<td></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

\[
\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92 \\
1/1.92 \approx 0.52 \\
1.3/1.92 \approx 0.68
\]

Final similarity score between query and document: \[\sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08\]

Questions?
tf-idf example: lnc.ltn

Query: “best car insurance”. Document: “car insurance auto insurance”.

<table>
<thead>
<tr>
<th>word</th>
<th>tf-raw</th>
<th>tf-wght</th>
<th>df</th>
<th>idf</th>
<th>weight</th>
<th>tf-raw</th>
<th>tf-wght</th>
<th>weight</th>
<th>n’lized</th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>best</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>1.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.04</td>
</tr>
<tr>
<td>insurance</td>
<td>2.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.04</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

\[ \sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92 \]

\[ \frac{1}{1.92} \approx 0.52 \]

\[ \frac{1.3}{1.92} \approx 0.68 \]

Final similarity score between query and document: \[ \sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08 \]

Questions?
tf-idf example: lnc.ltn

Query: “best car insurance”. Document: “car insurance auto insurance”.

<table>
<thead>
<tr>
<th>word</th>
<th>query</th>
<th>document</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tf-raw</td>
<td>tf-wght</td>
</tr>
<tr>
<td>auto</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>best</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>1.04</td>
<td></td>
</tr>
<tr>
<td>insurance</td>
<td>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.

\[
\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92 \\
1/1.92 \approx 0.52 \\
1.3/1.92 \approx 0.68
\]

Final similarity score between query and document: \( \sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08 \)

Questions?
tf-idf example: lnc.ltn

Query: “best car insurance”. Document: “car insurance auto insurance”.

<table>
<thead>
<tr>
<th>word</th>
<th>query</th>
<th>document</th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tf-raw  tf-wght  df  idf  weight</td>
<td>tf-raw  tf-wght  weight n’lized</td>
<td></td>
</tr>
<tr>
<td>auto</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>best</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>1.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>insurance</td>
<td>2.04</td>
<td></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

\[ \sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92 \]

\[ 1/1.92 \approx 0.52 \]

\[ 1.3/1.92 \approx 0.68 \]

Final similarity score between query and document: \[ \sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08 \]

Questions?
tf-idf example: lnc.ltn

Query: “best car insurance”. Document: “car insurance auto insurance”.

<table>
<thead>
<tr>
<th>word</th>
<th>query</th>
<th>document</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tf-raw</td>
<td>tf-wght</td>
</tr>
<tr>
<td>auto</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>best</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>1.04</td>
<td></td>
</tr>
<tr>
<td>insurance</td>
<td>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

\[
\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92 \\
1/1.92 \approx 0.52 \\
1.3/1.92 \approx 0.68
\]

Final similarity score between query and document: \[ \sum_{i} w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08 \]

Questions?
## tf-idf example: lnc.ltn

Query: “best car insurance”. Document: “car insurance auto insurance”.

<table>
<thead>
<tr>
<th>word</th>
<th>query</th>
<th>document</th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tf-raw</td>
<td>df</td>
<td>idf</td>
</tr>
<tr>
<td>auto</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>best</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>1.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>insurance</td>
<td>2.04</td>
<td></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.

\[
\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92 \\
1/1.92 \approx 0.52 \\
1.3/1.92 \approx 0.68
\]

Final similarity score between query and document: \( \sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08 \)

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

Query: “best car insurance”. Document: “car insurance auto insurance”.

<table>
<thead>
<tr>
<th>word</th>
<th>tf-raw</th>
<th>tf-wght</th>
<th>df</th>
<th>idf</th>
<th>weight</th>
<th>tf-raw</th>
<th>tf-wght</th>
<th>weight</th>
<th>n’lized</th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1.04</td>
<td>1.04</td>
<td>1</td>
<td>1.04</td>
<td>1.04</td>
<td>1.04</td>
<td>0 + 0.52</td>
</tr>
<tr>
<td>best</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1.04</td>
<td>1.04</td>
<td>1</td>
<td>1.04</td>
<td>1.04</td>
<td>1.04</td>
<td>0 + 0.52</td>
</tr>
<tr>
<td>car</td>
<td>1.04</td>
<td>1.04</td>
<td>1</td>
<td>1.04</td>
<td>1.04</td>
<td>1</td>
<td>1.04</td>
<td>1.04</td>
<td>1.04</td>
<td>0 + 0.52</td>
</tr>
<tr>
<td>insurance</td>
<td>2.04</td>
<td>2.04</td>
<td>1</td>
<td>1.04</td>
<td>2.04</td>
<td>2</td>
<td>2.04</td>
<td>2.04</td>
<td>2.04</td>
<td>2 + 1.28</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.

\[
\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92 \\
1/1.92 \approx 0.52 \\
1.3/1.92 \approx 0.68
\]

Final similarity score between query and document: \[\sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08\]

Questions?
**tf-idf example: inc.ltn**

Query: “best car insurance”. Document: “car insurance auto insurance”.

<table>
<thead>
<tr>
<th>word</th>
<th>query</th>
<th>document</th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tf-raw</td>
<td>tf-wght</td>
<td>df</td>
</tr>
<tr>
<td>auto</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>best</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>1.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>insurance</td>
<td>2.04</td>
<td></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

\[
\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92
\]

\[
1/1.92 \approx 0.52
\]

\[
1.3/1.92 \approx 0.68
\]

Final similarity score between query and document: \( \sum_i w_qi \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08 \)

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

Query: “best car insurance”. Document: “car insurance auto insurance”.

<table>
<thead>
<tr>
<th>word</th>
<th>tf-raw tf-wght</th>
<th>df</th>
<th>idf</th>
<th>weight</th>
<th>tf-raw tf-wght</th>
<th>weight</th>
<th>n’lized</th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>best</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>1.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>insurance</td>
<td>2.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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

\[
\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92 \\
1/1.92 \approx 0.52 \\
1.3/1.92 \approx 0.68
\]

Final similarity score between query and document: \( \sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08 \)

Questions?
**tf-idf example: lncltn**

Query: “best car insurance”. Document: “car insurance auto insurance”.

<table>
<thead>
<tr>
<th>word</th>
<th>query</th>
<th>document</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tf-raw</td>
<td>df</td>
</tr>
<tr>
<td>auto</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>best</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>1.04</td>
<td></td>
</tr>
<tr>
<td>insurance</td>
<td>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

\[
\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92
\]
\[
1/1.92 \approx 0.52
\]
\[
1.3/1.92 \approx 0.68
\]

Final similarity score between query and document: \[ \sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08 \]

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

Query: “best car insurance”. Document: “car insurance auto insurance”.

<table>
<thead>
<tr>
<th>word</th>
<th>query</th>
<th></th>
<th></th>
<th></th>
<th>document</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tf-raw</td>
<td>tf-wght</td>
<td>df</td>
<td>idf</td>
<td>weight</td>
<td>tf-raw</td>
<td>tf-wght</td>
<td>weight</td>
<td>n’lized</td>
<td></td>
</tr>
<tr>
<td>auto</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>best</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>car</td>
<td>1.04</td>
<td></td>
<td></td>
<td></td>
<td>1.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.68</td>
</tr>
<tr>
<td>insurance</td>
<td>2.04</td>
<td></td>
<td></td>
<td></td>
<td>2.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.68</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

\[
\sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92 \\
1/1.92 \approx 0.52 \\
1.3/1.92 \approx 0.68
\]

Final similarity score between query and document: \( \sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08 \)

Questions?
Summary: Ranked retrieval in the vector space model
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
How about a little Exercise(s)?
How about a little Exercise(s)?

- Group work is going well.
- Try 6.16, 6.17, 6.19 and 6.23 as a group
- ... What would you say if I based your scores on the weighted return values?
Take-away today

- **Ranking** search results: why it is important (as opposed to just presenting a set of unordered Boolean results)
- **Term frequency**: This is a key ingredient for ranking.
- **Tf-idf ranking**: best known traditional ranking scheme
- **Vector space model**: One of the most important formal models for information retrieval (along with Boolean and probabilistic models)
Recap

Why ranked retrieval?

Term frequency

tf-idf weighting

The vector space model

Resources

- Chapters 6 and 7 of IIR
- Resources at http://cislmu.org
  - 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)