IIR 2: The term vocabulary and postings lists

Hinrich Schütze

Center for Information and Language Processing, University of Munich

2014-04-09
Overview

1 Recap

2 Documents

3 Terms
   - General + Non-English
   - English

4 Skip pointers

5 Phrase queries
Definitions

- **Word** – A delimited string of characters as it appears in the text.
- **Term** – A “normalized” word (case, morphology, spelling etc); an equivalence class of words.
- **Token** – An instance of a word or term occurring in a document.
- **Type** – The same as a term in most cases: an equivalence class of tokens.
Normalization

- Need to “normalize” words in indexed text as well as query terms into the same form.
- Example: We want to match *U.S.A.* and *USA*
- We most commonly implicitly define *equivalence classes* of terms.
- Alternatively: do asymmetric expansion
  - *window* → window, windows
  - *windows* → Windows, windows
  - Windows (no expansion)
- More powerful, but less efficient
- Why don’t you want to put *window*, *Window*, *windows*, and *Windows* in the same equivalence class?
Tokenization: Recall construction of inverted index

- **Input:**

  Friends, Romans, countrymen.  
  So let it be with Caesar . . .

- **Output:**

  friend  
  roman  
  countryman  
  so  
  . . .

- Each token is a candidate for a postings entry.

- What are valid tokens to emit?
Exercises

In June, the dog likes to chase the cat in the barn. – How many word tokens? How many word types? Why tokenization is difficult – even in English. Tokenize: Mr. O’Neill thinks that the boys’ stories about Chile’s capital aren’t amusing.
Tokenization problems: One word or two? (or several)

- Hewlett-Packard
- State-of-the-art
- co-education
- the hold-him-back-and-drag-him-away maneuver
- data base
- San Francisco
- Los Angeles-based company
- cheap San Francisco-Los Angeles fares
- York University vs. New York University
Numbers

- 3/20/91
- 20/3/91
- Mar 20, 1991
- B-52
- 100.2.86.144
- (800) 234-2333
- 800.234.2333
- Older IR systems may not index numbers . . .
- . . . but generally it’s a useful feature.
- Google example
Outline

1 Recap

2 Documents

3 Terms
   - General + Non-English
   - English

4 Skip pointers

5 Phrase queries
Case folding

- Reduce all letters to lower case
- Even though case can be semantically meaningful
  - capitalized words in mid-sentence
  - MIT vs. mit
  - Fed vs. fed
  - ...

- It’s often best to lowercase everything since users will use lowercase regardless of correct capitalization.
Stop words

- stop words = extremely common words which would appear to be of little value in helping select documents matching a user need
- Examples: a, an, and, are, as, at, be, by, for, from, has, he, in, is, it, its, of, on, that, the, to, was, were, will, with
- Stop word elimination used to be standard in older IR systems.
- But you need stop words for phrase queries, e.g. “King of Denmark”
- Most web search engines index stop words.
More equivalence classing

- **Soundex**: IIR 3 (phonetic equivalence, Muller = Mueller)
- **Thesauri**: IIR 9 (semantic equivalence, car = automobile)
Lemmatization

- Reduce inflectional/variant forms to base form
- Example: $am, are, is \rightarrow be$
- Example: $car, cars, car’s, cars’ \rightarrow car$
- Example: $the\ boy’s\ cars\ are\ different\ colors \rightarrow the\ boy\ car\ be\ different\ color$
- Lemmatization implies doing “proper” reduction to dictionary headword form (the lemma).
- Inflectional morphology ($cutting \rightarrow cut$) vs. derivational morphology ($destruction \rightarrow destroy$)
**Stemming**

- Definition of stemming: Crude heuristic process that **chops off the ends of words** in the hope of achieving what “principled” lemmatization attempts to do with a lot of linguistic knowledge.
- Language dependent
- Often inflectional and derivational
- Example for derivational: *automate, automatic, automation* all reduce to *automat*
Porter algorithm

- Most common algorithm for stemming English
- Results suggest that it is at least as good as other stemming options
- Conventions + 5 phases of reductions
- Phases are applied sequentially
- Each phase consists of a set of commands.
  - Sample command: Delete final *ement* if what remains is longer than 1 character
    - replacement → replac
    - cement → cement
- Sample convention: Of the rules in a compound command, select the one that applies to the longest suffix.
Porter stemmer: A few rules

<table>
<thead>
<tr>
<th>Rule</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSES → SS</td>
<td>caresses → caress</td>
</tr>
<tr>
<td>IES → I</td>
<td>ponies → poni</td>
</tr>
<tr>
<td>SS → SS</td>
<td>caress → caress</td>
</tr>
<tr>
<td>S →</td>
<td>cats → cat</td>
</tr>
</tbody>
</table>
Three stemmers: A comparison

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation.

Porter stemmer: such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation.

Lovins stemmer: such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation.

Paice stemmer: such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation.
Does stemming improve effectiveness?

- In general, stemming increases effectiveness for some queries, and decreases effectiveness for others.
- Queries where stemming is likely to help: [tartan sweaters], [sightseeing tour san francisco]
- (equivalence classes: \{sweater, sweaters\}, \{tour, tours\})
- Porter Stemmer equivalence class \textit{oper} contains all of \textit{operate operating operates operation operative operatives operational}.
- Queries where stemming hurts: [operational AND research], [operating AND system], [operative AND dentistry]
Exercise: What does Google do?

- Stop words
- Normalization
- Tokenization
- Lowercasing
- Stemming
- Non-latin alphabets
- Umlauts
- Compounds
- Numbers
Overview

1 Recap

2 Why ranked retrieval?

3 Term frequency

4 tf-idf weighting

5 The vector space model
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**: Important formal model 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

- 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.
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

- 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

- How can we accomplish a *relevance ranking* of the documents 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”.
- Sort documents according to scores
Query-document matching scores

- How do we compute the score of a query-document pair?
- If no query term occurs in the document: score should be 0.
- The more frequent a query term in the document, the higher the score.
- The more query terms occur in the document, the higher the score.
- We will look at a number of alternatives for doing this.
Take 1: Jaccard coefficient

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

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

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

- $JACCARD(A, A) = 1$
- $JACCARD(A, B) = 0$ if $A \cap B = 0$
- $A$ and $B$ don’t have to be the same size.
- Always assigns a number between 0 and 1.
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?

- 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 $|A \cap B|/\sqrt{|A \cup B|}$ (cosine) . . .
- . . . instead of $|A \cap B|/|A \cup B|$ (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>ANTHONY</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></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>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>
<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>1</td>
<td>0</td>
<td>1</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>
<td></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</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>Worser</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

- 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
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.
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 \( d \):

\[
tf\text{-matching-score}(q, d) = \sum_{t \in q \cap d} (1 + \log tf_{t,d})
\]

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

- 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

- 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 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

- $df_t$ 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

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

- 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
- 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”).
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
- Alternative names: tf.idf, tf \times idf
Summary: tf-idf

- Assign a tf-idf weight for each term \( t \) in each document \( d \):
  \[
  w_{t,d} = (1 + \log \text{tf}_{t,d}) \cdot \log \frac{N}{\text{df}_t}
  \]
- 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 $df$?
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 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 1 0 0 0 1</td>
<td></td>
<td>0 0 0 0 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BRUTUS</strong></td>
<td>1 1 0 1 0 0</td>
<td></td>
<td>1 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CAESAR</strong></td>
<td>1 1 0 1 1 1</td>
<td></td>
<td>1 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CALPURNIA</strong></td>
<td>0 1 0 0 0 0</td>
<td></td>
<td>0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CLEOPATRA</strong></td>
<td>1 0 0 0 0 0</td>
<td></td>
<td>0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MERCY</strong></td>
<td>1 0 1 1 1 1</td>
<td></td>
<td>1 1 1 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>WORSER</strong></td>
<td>1 0 1 1 1 0</td>
<td></td>
<td>1 1 1 1 0</td>
<td></td>
<td></td>
<td></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>Othello</th>
<th>Macbeth</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anthony</td>
<td>157 73 0 0 0 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Brutus</td>
<td>4 157 2 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Caesar</td>
<td>232 227 2 1 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0 10 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cleopatra</td>
<td>57 0 0 0 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mercy</td>
<td>2 0 3 8 5 8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Worse</td>
<td>2 0 1 1 1 5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each document is now represented as a count vector $\in \mathbb{N}^{|V|}$. 
<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>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>Worsener</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 $\in \mathbb{R}^{|V|}$. 
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

- **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 \approx \text{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?

- First cut: (negative) 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 $d'$. $d'$ 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 cosine(query, document) in increasing order.

Cosine is a monotonically decreasing function of the angle for the interval \([0^\circ, 180^\circ]\).
Cosine
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

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

- \(q_i\) 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 for normalized vectors

- For normalized vectors, the cosine is equivalent to the dot product or scalar product.
- \( \cos(\vec{q}, \vec{d}) = \vec{q} \cdot \vec{d} = \sum_i q_i \cdot d_i \)
  - (if \( \vec{q} \) and \( \vec{d} \) are length-normalized).
Cosine similarity illustrated
Cosine: Example

How similar are these novels? SaS:
Sense and Sensibility
PaP:
Pride and Prejudice
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>
## 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>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>

<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>

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

#### log frequency weighting

<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>

#### log frequency weighting & cosine normalization

<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** \( \cos(SaS, PaP) > \cos(SaS, WH) \)?
## 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;t,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;t,d&lt;/sub&gt;)</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 tf_{t,d}}{\max_{t}(tf_{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 } tf_{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(tf_{t,d})}{1 + \log(\text{ave}<em>{t \in d}(tf</em>{t,d}))} )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Best known combination of weighting options Default: no

weighting
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 idf-weight the document?
- Example query: “best car insurance”
- Example document: “car insurance auto insurance”
**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>tf-wght</td>
<td>df</td>
</tr>
<tr>
<td>auto</td>
<td>0</td>
<td>0</td>
<td>5000</td>
</tr>
<tr>
<td>best</td>
<td>1</td>
<td>1</td>
<td>50000</td>
</tr>
<tr>
<td>car</td>
<td>1</td>
<td>1</td>
<td>10000</td>
</tr>
<tr>
<td>insurance</td>
<td>1</td>
<td>1</td>
<td>1000</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 \text{ Final similarity score between query and document: } \sum_i w_{qi} \cdot w_{di} = 0 + 0 + 1.04 + 2.04 = 3.08 \text{ Questions?}
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
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**: Important formal model for information retrieval (along with Boolean and probabilistic models)
Resources

- Chapters 6 and 7 of IIR
- Resources at http://cis.lmu.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)