CS 572: Information Retrieval

Lecture 5: Term Weighting and Ranking

Acknowledgment: Some slides in this lecture are adapted from Chris Manning (Stanford) and Doug Oard (Maryland)
Lecture Plan

• Skip for now index optimization:
  – Distributed, tiered, caching: \(\rightarrow\) return to it later

• Term weighting

• Vector space model of IR
Relevance

• **Relevance** relates a **topic** and a document
  – Duplicates are equally relevant, by definition
  – Constant over time and across users

• **Pertinence** relates a **task** and a document
  – Accounts for quality, complexity, language, ...

• **Utility** relates a **user** and a document
  – Accounts for prior knowledge (e.g., search session)

• **We want utility, but (for now) we get relevance**
Advantages of Ranked Retrieval

- Closer to the way people think
  - Some documents are better than others

- Enriches browsing behavior
  - Decide how far down the list to go as you read it

- Allows more flexible queries
  - Long and short queries can produce useful results
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”.
Attempt 1: Linear zone combinations

• First generation of scoring methods: use a linear combination of Booleans:

\[
\text{Score} = 0.6*<\text{sorting in Title}> + 0.3*<\text{sorting in Abstract}> + 0.05*<\text{sorting in Body}> + 0.05*<\text{sorting in Boldface}>
\]

– Each expression such as <\text{sorting in Title}> takes on a value in \{0,1\}.
– Then the overall score is in [0,1].

For this example the scores can only take on a finite set of values – what are they?
Linear zone combinations

• The expressions between <> on the last slide could be *any* Boolean query

• Who generates the Score expression (with weights such as 0.6 etc.)?
  – In uncommon cases – the user through the UI
  – Most commonly, a *query parser* that takes the user’s Boolean query and runs it on the indexes for each zone
  – Weights determined from user studies and hard-coded into the query parser.
For query *bill OR rights* suppose that we retrieve the following docs from the various zone indexes:

<table>
<thead>
<tr>
<th>Author</th>
<th>bill</th>
<th>1 → 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rights</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>bill</td>
<td>3 → 5 → 8</td>
</tr>
<tr>
<td></td>
<td>rights</td>
<td>3 → 5 → 9</td>
</tr>
<tr>
<td>Body</td>
<td>bill</td>
<td>1 → 2 → 5 → 9</td>
</tr>
<tr>
<td></td>
<td>rights</td>
<td>3 → 5 → 8 → 9</td>
</tr>
</tbody>
</table>

Semantics of “OR”: both appearing are “better” than only 1 term?
General idea

• We are given a weight vector whose components sum up to 1.
  – There is a weight for each zone/field.

• Given a Boolean query, we assign a score to each doc by adding up the weighted contributions of the zones/fields.
Index support for zone combinations

- In the simplest version we have a separate inverted index for each zone
- Variant: have a single index with a separate dictionary entry for each term and zone
- E.g., $\text{bill.author}$
  
  $\text{bill.title}$
  
  $\text{bill.body}$
Zone combinations index

• The above scheme is still wasteful: each term is potentially replicated for each zone
• In a slightly better scheme, we encode the zone in the postings:

```
bill  1.author, 1.body  2.author, 2.body  3.title
```

• At query time, accumulate contributions to the total score of a document from the various postings

zone names get compressed.
Score accumulation ex:

- As we walk the postings for the query **bill OR rights**, we accumulate scores for each doc in a linear merge as before.
- Note: we get both **bill** and **rights** in the Title field of doc 3, but score it no higher.
The Perfect Query Paradox

• Every information need has a perfect doc set
  – If not, there would be no sense doing retrieval
• Almost every document set has a perfect query
  – AND every word to get a query for document 1
  – Repeat for each document in the set
  – OR every document query to get the set query
• But users find Boolean query formulation hard
  – They get too much, too little, useless stuff, ...
Why Boolean Retrieval Fails

• Natural language is way more complex
  – She saw the man on the hill with a telescope

• AND “discovers” nonexistent relationships
  – Terms in different paragraphs, chapters, ...

• Guessing terminology for OR is hard
  – good, nice, excellent, outstanding, awesome, ...

• Guessing terms to exclude is even harder!
  – Democratic party, party to a lawsuit, ...
Problem with Boolean search: feast or famine

• Boolean queries often result in either too few (=0) or too many (1000s) results.
• Query 1: “standard user dlink 650” → 200,000 hits
• Query 2: “standard user dlink 650 no card found”: 0 hits
• It takes a lot of skill to come up with a query that produces a manageable number of hits.
  – AND gives too few; OR gives too many
Boolean IR: Strengths and Weaknesses

• Strong points
  – Accurate, **if you know the right strategies**
  – Efficient for the computer

• Weaknesses
  – Often results in too many documents, or none
  – Users must learn Boolean logic
  – Sometimes finds relationships that don’t exist
  – Words can have many meanings
  – Choosing the right words is sometimes hard
Ranked Retrieval Paradigm

• Exact match retrieval often gives useless sets
  – No documents at all, or way too many documents

• Query reformulation is one “solution”
  – Manually add or delete query terms

• “Best-first” ranking can be superior
  – Select every document within reason
  – Put them in order, with the “best” ones first
  – Display them one screen at a time
Feast or famine: not a problem in ranked retrieval

- When a system produces a ranked result set, large result sets are not an issue
  - 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”
Free Text Queries

- We just scored the Boolean query *bill OR rights*
- Most users more likely to type *bill rights* or *bill of rights*
  - How do we interpret these “free text” queries?
  - No Boolean connectives
  - Of several query terms some may be missing in a doc
  - Only some query terms may occur in the title, etc.
Incidence matrices

- Recall: Document (or a zone in it) is binary vector $X$ in $\{0, 1\}^v$
  - Query is a vector
- Score: Overlap measure:

$$\left| X \cap Y \right|$$

<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>
Example

- On the query *ides of march*, Shakespeare’s *Julius Caesar* has a score of 3
- All other Shakespeare plays have a score of 2 (because they contain *march*) or 1
- Thus in a rank order, *Julius Caesar* would be 1st
Overlap matching

What’s wrong with the overlap measure?

It doesn’t consider:

- Term frequency in document
- Term rarity in collection (document mention frequency)
  - *of* is more common than *ides* or *march*
- Length of documents
  - (And queries: score not normalized)
Overlap matching

- One can normalize in various ways:
  - Jaccard coefficient:
    \[
    \frac{|X \cap Y|}{|X \cup Y|}
    \]
  - Cosine measure:
    \[
    \frac{|X \cap Y|}{\sqrt{|X| \times |Y|}}
    \]

- What documents would score **highest** using Jaccard against a typical query?
  - Does the cosine measure fix this problem?
Term Weighting: Empirical Motivation

• During retrieval:
  – Find the relevant postings based on query terms
  – Manipulate the postings based on the query
  – Return appropriate documents

• Example with Boolean queries

• What about postings for “unimportant” terms?
  – “a”, “the”, “not”, ...?
Extreme conditions create rare Antarctic clouds

SYDNEY (Reuters) - Rare, mother-of-pearl colored clouds caused by extreme weather conditions above Antarctica are a possible indication of global warming, Australian scientists said on Tuesday.

Known as nacreous clouds, the spectacular formations showing delicate wisps of colors were photographed in the sky over an Australian meteorological base at Mawson Station on July 25.
# Reuters RCV1 statistics

<table>
<thead>
<tr>
<th>symbol</th>
<th>statistic</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>documents</td>
<td>800,000</td>
</tr>
<tr>
<td>L</td>
<td>avg. # <strong>tokens</strong> per doc</td>
<td>200</td>
</tr>
<tr>
<td>M</td>
<td><strong>terms</strong> (= word <strong>types</strong>)</td>
<td>400,000</td>
</tr>
<tr>
<td></td>
<td>avg. # bytes per <strong>token</strong></td>
<td>6 (incl. spaces/punct.)</td>
</tr>
<tr>
<td></td>
<td>avg. # bytes per <strong>token</strong></td>
<td>4.5 (w/out spaces/punct.)</td>
</tr>
<tr>
<td></td>
<td>avg. # bytes per <strong>term</strong></td>
<td>7.5</td>
</tr>
<tr>
<td></td>
<td>non-positional postings</td>
<td>100,000,000</td>
</tr>
</tbody>
</table>

**Note: 4.5 bytes per word **token** vs. 7.5 bytes per word **type****
Term Frequency: Zipf’s Law

- George Kingsley Zipf (1902-1950) observed that for many frequency distributions, the $n$th most frequent event is related to its frequency as:

\[ f \cdot r = c \quad \text{or} \quad f = \frac{c}{r} \]

$f = \text{frequency}$
$r = \text{rank}$
$c = \text{constant}$
Zipfian Distribution

![Graph showing frequency and rank relationship](image)

- **Graph 1**: Frequency vs. Rank
- **Graph 2**: Log(Frequency) vs. Log(Rank)

**Zipf's Law**: The frequency of an item is inversely proportional to its rank in a frequency table.
Zipf’s law for Reuters RCV1
Zipfian Distribution

• Key points:
  – A few elements occur very frequently
  – A medium number of elements have medium frequency
  – Many elements occur very infrequently

• Why do we care?
  – English word frequencies follow Zipf’s Law
### Word Frequency in English

#### Frequency of 50 most common words in English
(sample of 19 million words)

<table>
<thead>
<tr>
<th>Word</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>1130021</td>
</tr>
<tr>
<td>of</td>
<td>547311</td>
</tr>
<tr>
<td>to</td>
<td>516635</td>
</tr>
<tr>
<td>a</td>
<td>464736</td>
</tr>
<tr>
<td>in</td>
<td>390819</td>
</tr>
<tr>
<td>and</td>
<td>387703</td>
</tr>
<tr>
<td>that</td>
<td>204351</td>
</tr>
<tr>
<td>for</td>
<td>199340</td>
</tr>
<tr>
<td>is</td>
<td>152483</td>
</tr>
<tr>
<td>said</td>
<td>148302</td>
</tr>
<tr>
<td>it</td>
<td>134323</td>
</tr>
<tr>
<td>on</td>
<td>121173</td>
</tr>
<tr>
<td>by</td>
<td>118863</td>
</tr>
<tr>
<td>as</td>
<td>109135</td>
</tr>
<tr>
<td>at</td>
<td>101779</td>
</tr>
<tr>
<td>mr</td>
<td>101679</td>
</tr>
<tr>
<td>with</td>
<td>101210</td>
</tr>
<tr>
<td>from</td>
<td>96900</td>
</tr>
<tr>
<td>he</td>
<td>94585</td>
</tr>
<tr>
<td>million</td>
<td>93515</td>
</tr>
<tr>
<td>year</td>
<td>90104</td>
</tr>
<tr>
<td>its</td>
<td>86774</td>
</tr>
<tr>
<td>be</td>
<td>85588</td>
</tr>
<tr>
<td>was</td>
<td>83398</td>
</tr>
<tr>
<td>company</td>
<td>83070</td>
</tr>
<tr>
<td>an</td>
<td>76974</td>
</tr>
<tr>
<td>has</td>
<td>74405</td>
</tr>
<tr>
<td>are</td>
<td>74097</td>
</tr>
<tr>
<td>have</td>
<td>73132</td>
</tr>
<tr>
<td>but</td>
<td>71887</td>
</tr>
<tr>
<td>will</td>
<td>71494</td>
</tr>
<tr>
<td>say</td>
<td>66807</td>
</tr>
<tr>
<td>new</td>
<td>64456</td>
</tr>
<tr>
<td>share</td>
<td>63925</td>
</tr>
<tr>
<td>or</td>
<td>54958</td>
</tr>
<tr>
<td>about</td>
<td>53713</td>
</tr>
<tr>
<td>market</td>
<td>52110</td>
</tr>
<tr>
<td>they</td>
<td>51359</td>
</tr>
<tr>
<td>this</td>
<td>50933</td>
</tr>
<tr>
<td>would</td>
<td>50828</td>
</tr>
<tr>
<td>you</td>
<td>49281</td>
</tr>
<tr>
<td>which</td>
<td>48273</td>
</tr>
<tr>
<td>bank</td>
<td>47940</td>
</tr>
<tr>
<td>stock</td>
<td>47401</td>
</tr>
<tr>
<td>trade</td>
<td>47310</td>
</tr>
<tr>
<td>his</td>
<td>47116</td>
</tr>
<tr>
<td>more</td>
<td>46244</td>
</tr>
<tr>
<td>who</td>
<td>42142</td>
</tr>
<tr>
<td>one</td>
<td>41635</td>
</tr>
<tr>
<td>their</td>
<td>40910</td>
</tr>
</tbody>
</table>
Does it fit Zipf’s Law?

The following shows $rf*1000/n$

$r$ is the rank of word $w$ in the sample
$f$ is the frequency of word $w$ in the sample
$n$ is the total number of word occurrences in the sample

<table>
<thead>
<tr>
<th>Word</th>
<th>Rank</th>
<th>Frequency</th>
<th>Total Words</th>
<th>Rank</th>
<th>Frequency</th>
<th>Total Words</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>59</td>
<td>59</td>
<td>92</td>
<td>or</td>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>of</td>
<td>58</td>
<td>82</td>
<td>95</td>
<td>about</td>
<td>102</td>
<td>102</td>
</tr>
<tr>
<td>to</td>
<td>82</td>
<td>98</td>
<td>98</td>
<td>market</td>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>a</td>
<td>98</td>
<td>103</td>
<td>100</td>
<td>they</td>
<td>103</td>
<td>103</td>
</tr>
<tr>
<td>in</td>
<td>103</td>
<td>103</td>
<td>100</td>
<td>this</td>
<td>105</td>
<td>105</td>
</tr>
<tr>
<td>and</td>
<td>122</td>
<td>75</td>
<td>105</td>
<td>would</td>
<td>107</td>
<td>107</td>
</tr>
<tr>
<td>that</td>
<td>75</td>
<td>72</td>
<td>105</td>
<td>you</td>
<td>106</td>
<td>106</td>
</tr>
<tr>
<td>for</td>
<td>84</td>
<td>78</td>
<td>106</td>
<td>which</td>
<td>107</td>
<td>107</td>
</tr>
<tr>
<td>is</td>
<td>72</td>
<td>78</td>
<td>105</td>
<td>bank</td>
<td>109</td>
<td>109</td>
</tr>
<tr>
<td>said</td>
<td>78</td>
<td>77</td>
<td>112</td>
<td>stock</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>it</td>
<td>78</td>
<td>81</td>
<td>114</td>
<td>trade</td>
<td>112</td>
<td>112</td>
</tr>
<tr>
<td>on</td>
<td>77</td>
<td>80</td>
<td>117</td>
<td>his</td>
<td>114</td>
<td>114</td>
</tr>
<tr>
<td>by</td>
<td>81</td>
<td>80</td>
<td>117</td>
<td>more</td>
<td>114</td>
<td>114</td>
</tr>
<tr>
<td>as</td>
<td>80</td>
<td>80</td>
<td>117</td>
<td>who</td>
<td>106</td>
<td>106</td>
</tr>
<tr>
<td>at</td>
<td>80</td>
<td>86</td>
<td>112</td>
<td>one</td>
<td>107</td>
<td>107</td>
</tr>
<tr>
<td>mr</td>
<td>86</td>
<td>91</td>
<td>114</td>
<td>their</td>
<td>108</td>
<td>108</td>
</tr>
<tr>
<td>with</td>
<td>91</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Explanation for Zipfian distributions

• Zipf’s own explanation ("least effort" principle):
  – Speaker’s goal is to minimise effort by using a few distinct words as frequently as possible
  – Hearer’s goal is to maximise clarity by having as large a vocabulary as possible

• Update: Zipfian distribution describes **phrases** better than words (worth a Nature paper!?!):
  [http://www.nature.com/articles/srep12209](http://www.nature.com/articles/srep12209)
Issues with Jaccard for scoring

- It doesn’t consider *term frequency* (how many times a term occurs in a document)
- Rare terms in a collection are more informative than frequent terms. Jaccard doesn’t consider this information
- We need a more sophisticated way of normalizing for length
- Later in this lecture, we’ll use $|A \cap B| / \sqrt{|A \cup B|}$ instead of $|A \cap B| / |A \cup B|$ (Jaccard) for length normalization.
Term-document **count matrices**

- 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.

- In a sense, this is a step back: The positional index was able to distinguish these two documents.

- We will look at “recovering” positional information later in this course.

- For now: bag of words model
Term frequency $tf$

- The term frequency $tf_{t,d}$ of term $t$ in document $d$ is defined as the number of times that $t$ occurs in $d$.
- We want to use $tf$ when computing query-document match scores. But how?
- Raw term frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

NB: frequency = count in IR
Log-frequency tf weighting

- The log frequency weight of term \( t \) in \( d \) is

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

- \( 0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4, \) etc.

- Score for a document-query pair: sum over terms \( t \) in both \( q \) and \( d \):

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

- The score is 0 if none of the query terms is present in the document.
Weighting should depend on the term overall

• Which of these tells you more about a doc?
  – 10 occurrences of *hernia*?
  – 10 occurrences of *the*?

• Would like to attenuate weights of common terms
  – But what is “common”?

• Can use **collection frequency** *(cf)*
  – The total number of occurrences of the term in the entire collection of documents
Document frequency $df$

- Document frequency ($df$) may be better:
- $df =$ number of docs in the corpus containing the term

<table>
<thead>
<tr>
<th>Word</th>
<th>$cf$</th>
<th>$df$</th>
</tr>
</thead>
<tbody>
<tr>
<td>try</td>
<td>10422</td>
<td>8760</td>
</tr>
<tr>
<td>insurance</td>
<td>10440</td>
<td>3997</td>
</tr>
</tbody>
</table>

- So how do we make use of $df$?
Document frequency

- Rare terms are more informative than frequent terms
  - the, a, of, ...

- 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*

- We want a high weight for rare terms like *arachnocentric*.
**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 $t$
  - $df_t \leq N$

- 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.
idf example, suppose $N = 1$ million

<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>1,000</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>

$$idf_t = \log \left( \frac{N}{df_t} \right)$$

There is one idf value for each term $t$ in a collection.
Effect of idf on ranking

• Does idf have an effect on ranking for one-term queries, like
  – iPhone
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.

\[ w_{t,d} = (1 + \log \text{tf}_{t,d}) \times \log_{10}(N / \text{df}_t) \]

- Best known weighting scheme in information retrieval
  - 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
Final ranking of documents for a query

\[ \text{Score}(q,d) = \sum_{t \in q \cap d} \text{tf}. \text{idf}_{t,d} \]
Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$
Documents as vectors

• 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

• Very sparse vectors - most entries are zero.
Why turn docs into vectors?

• First application: Query-by-example
  – Given a doc D, find others “like” it.
  – What are some applications?

• Now that D is a vector, find vectors (docs) “near” it.
Postulate: Documents that are “close together” in the vector space are about the same things.
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 ≈ inverse of distance

• Recall: We do this because we want to get away from the all-or-nothing 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.
Desiderata for proximity

• If $d_1$ is near $d_2$, then $d_2$ is near $d_1$ (bijection)

• If $d_1$ near $d_2$, and $d_2$ near $d_3$, then $d_1$ is not far from $d_3$ (transitivity)

• No doc is closer to $d$ than $d$ itself. (idempotence)
Why distance is a bad idea

The Euclidean distance between $q$ and $d_2$ is large even though the distribution of terms in the query $q$ and the distribution of terms in the document $d_2$ 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 \( \cos\text{ine}(query, document) \)

• 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:

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

• 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
The cosine similarity of two vectors \( \mathbf{q} \) and \( \mathbf{d} \) is given by:

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

where:

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

\( \cos(\mathbf{q}, \mathbf{d}) \) is the cosine similarity of \( \mathbf{q} \) and \( \mathbf{d} \) ... or, equivalently, the cosine of the angle between \( \mathbf{q} \) and \( \mathbf{d} \).
Cosine for length-normalized vectors

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

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

for \( q, d \) length-normalized.
Cosine Similarity Example

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

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

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

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

\[ \theta \]

POOR

RICH
Cosine similarity amongst 3 documents

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

<table>
<thead>
<tr>
<th>term</th>
<th>SaS</th>
<th>PaP</th>
<th>WH</th>
</tr>
</thead>
<tbody>
<tr>
<td>affection</td>
<td>115</td>
<td>58</td>
<td>20</td>
</tr>
<tr>
<td>jealous</td>
<td>10</td>
<td>7</td>
<td>11</td>
</tr>
<tr>
<td>gossip</td>
<td>2</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>wuthering</td>
<td>0</td>
<td>0</td>
<td>38</td>
</tr>
</tbody>
</table>

**Term frequencies (counts)**

Note: To simplify this example, we don’t do idf weighting.
3 documents example (cont’d)

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>2.00</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>

After length 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</td>
<td>0.405</td>
</tr>
<tr>
<td>wuthering</td>
<td>0</td>
<td>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 cosine scores

\textbf{CosineScore}(q)
1 \hspace{1em} \textit{float} \textit{Scores}[N] = 0
2 \hspace{1em} \textit{float} \textit{Length}[N]
3 \hspace{1em} \textbf{for each} \hspace{1em} \textit{query term} \textit{t}
4 \hspace{1em} \textbf{do} \hspace{1em} \textit{calculate} \textit{w}_{t,q} \hspace{1em} \textit{and} \hspace{1em} \textit{fetch} \hspace{1em} \textit{postings list for} \hspace{1em} \textit{t}
5 \hspace{1em} \hspace{1em} \textbf{for each} \hspace{1em} \textit{pair}(d, \textit{tf}_{t,d}) \hspace{1em} \textit{in} \hspace{1em} \textit{postings list}
6 \hspace{1em} \hspace{1em} \hspace{1em} \hspace{1em} \textbf{do} \hspace{1em} \textit{Scores}[d] + = \textit{w}_{t,d} \times \textit{w}_{t,q}
7 \hspace{1em} \textit{Read the array} \textit{Length}
8 \hspace{1em} \textbf{for each} \hspace{1em} \textit{d}
9 \hspace{1em} \textbf{do} \hspace{1em} \textit{Scores}[d] = \textit{Scores}[d]/\textit{Length}[d]
10 \hspace{1em} \textbf{return} \hspace{1em} \textit{Top K components of} \textit{Scores}[]
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>tf_{t,d}</td>
<td>n (no)</td>
</tr>
<tr>
<td>l (logarithm)</td>
<td>1 + \log(tf_{t,d})</td>
<td>t (idf)</td>
</tr>
<tr>
<td>a (augmented)</td>
<td>0.5 + \frac{0.5 \times tf_{t,d}}{\max_t(tf_{t,d})}</td>
<td>p (prob idf)</td>
</tr>
<tr>
<td>b (boolean)</td>
<td>1 \text{ if } tf_{t,d} &gt; 0, 0 \text{ otherwise}</td>
<td>max{0, \log \frac{N-\text{df}_t}{\text{df}_t}}</td>
</tr>
<tr>
<td>L (log ave)</td>
<td>\frac{1 + \log(tf_{t,d})}{1 + \log(\text{ave}<em>{t \in d}(tf</em>{t,d}))}</td>
<td>1/CharLength^{\alpha}, \alpha &lt; 1</td>
</tr>
</tbody>
</table>

Columns headed ‘n’ are acronyms for weight schemes.

Why is the base of the log in idf immaterial?
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 **ddd.qqq**, using the acronyms from the previous table

• A very standard weighting scheme is: **Lnc.Ltc**
  – Document: logarithmic tf (**L** as first character), **no** idf and **yes** cosine normalization
  – Query: logarithmic tf (**L** in leftmost column), **idf** (**t** in second column), cosine normalization ...
tf-idf example: Lnc.Ltc

Document: *car insurance auto insurance*
Query: *best car insurance*

<table>
<thead>
<tr>
<th>Term</th>
<th>Query</th>
<th>Document</th>
<th>Prod</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>tf-raw</td>
<td>tf-wt</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>

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

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

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

$1/27/2016$

CS572: Information Retrieval. Spring 2016
Summary – vector space ranking

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

- MRS Ch 4, Ch 6.1 – 6.4