Ranked retrieval

Nisheeth
Ranked retrieval

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

• Good for expert users with precise understanding of their needs and the collection.
  – Also good for applications: Applications can easily consume 1000s of results.

• Not good for the majority of users.
  – Writing Boolean queries is hard
Problem with Boolean search: feast or famine

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

• Rather than a set of documents satisfying a query expression, in *ranked retrieval*, the system returns an ordering over the (top) documents in the collection for a query

• **Free text queries:** Rather than a query language of operators and expressions, the user’s query is just one or more words in a human language

• ** Ranked list of results:** No more feast or famine
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”.

Ch. 6
Query-document matching scores

- We need a way of assigning a score to a query/document pair
- If the query term does not occur in the document: score should be 0
- The more frequent the query term in the document, the higher the score (should be)
Jaccard coefficient

- \( \text{jaccard}(A,B) = \frac{|A \cap B|}{|A \cup B|} \)
- \( \text{jaccard}(A,A) = 1 \)
- \( \text{jaccard}(A,B) = 0 \) if \( A \cap B = 0 \)
- Always assigns a number between 0 and 1.

<table>
<thead>
<tr>
<th>Attributes occurring in both samples</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes occurring in sample ( X ) only</td>
<td>1</td>
</tr>
<tr>
<td>Attributes occurring in sample ( Y ) only</td>
<td>2</td>
</tr>
</tbody>
</table>

\[ = \frac{4}{(4+1+2)} \]

Jaccard's Similarity Index \( \rightarrow 0.57 \)
Issues with Jaccard for scoring

• Privileges shorter documents
  – We need a more sophisticated way of normalizing for length \( \frac{|A \cap B|}{\sqrt{|A \cup B|}} \)

• It doesn’t consider term frequency
  – how many times a term occurs in a document

• Does not account for term informativeness
  – How important is the term in the document
Accounting for term frequency

<table>
<thead>
<tr>
<th></th>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Brutus</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>worser</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$
Term frequency tf

- The term frequency $t_{t,d}$ 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.
Log-frequency weighting

• The log frequency weight of term $t$ in $d$ is
  
  $$w_{t,d} = \begin{cases} 
  1 + \log_{10} tf_{t,d}, & \text{if } tf_{t,d} > 0 \\
  0, & \text{otherwise}
  \end{cases}$$

• $0 \rightarrow 0$, $1 \rightarrow 1$, $2 \rightarrow 1.3$, $10 \rightarrow 2$, $1000 \rightarrow 4$, etc.

• Score for a document-query pair: sum over terms $t$ in both $q$ and $d$:
  
  $$\text{score} = \sum_{t \in q \cap d} (1 + \log tf_{t,d})$$

• The score is 0 if none of the query terms is present in the document.
Document frequency

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

• Frequent terms are less informative than rare terms
• Consider a query term that is frequent in the collection (e.g., *high*, *increase*, *line*)
• A document containing such a term is more likely to be relevant than a document that doesn’t
• But it’s not a sure indicator of relevance.
  – How/when will it break?
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 $\log \left( \frac{N}{df_t} \right)$ 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>

$idf_t = \log_{10} \left( \frac{N}{df_t} \right)$

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

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

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

• Best known weighting scheme in information retrieval

• Increases with the number of occurrences within a document

• Increases with the rarity of the term in the collection
Effect of idf on ranking

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

• idf has no effect on ranking one term queries
  – idf affects the ranking of documents for queries with at least two terms
  – For the query capricious person, idf weighting makes occurrences of capricious count for much more in the final document ranking than occurrences of person.
Score for a document given a query

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

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

<table>
<thead>
<tr>
<th>Term frequency</th>
<th>Document frequency</th>
<th>Normalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>n (natural)</td>
<td>n (no)</td>
<td>n (none)</td>
</tr>
<tr>
<td>$t_f, t_d$</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

| Columns headed ‘n’ are acronyms for weight schemes. |

<table>
<thead>
<tr>
<th>Term frequency</th>
<th>Document frequency</th>
<th>Normalization</th>
</tr>
</thead>
<tbody>
<tr>
<td>l (logarithm)</td>
<td>1 + log($t_f, t_d$)</td>
<td>1</td>
</tr>
<tr>
<td>a (augmented)</td>
<td>$0.5 + \frac{0.5 \times t_f, t_d}{\max(t_f, t_d)}$</td>
<td>$\frac{1}{\sqrt{w_1^2 + w_2^2 + \ldots + w_M^2}}$</td>
</tr>
<tr>
<td>b (boolean)</td>
<td>$\begin{cases} 1 &amp; \text{if } t_f, t_d &gt; 0 \ 0 &amp; \text{otherwise} \end{cases}$</td>
<td>$1/u$</td>
</tr>
<tr>
<td>L (log ave)</td>
<td>$\frac{1 + \log(t_f, t_d)}{1 + \log(\text{ave}_{t \in d}(t_f, t_d))}$</td>
<td>$1/CharLength^{\alpha}$, $\alpha &lt; 1$</td>
</tr>
</tbody>
</table>
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: Inc.ltc
    • Document: logarithmic tf (l as first character), no idf and cosine normalization
    • Query: logarithmic tf (l in leftmost column), idf (t in second column), no normalization ...
### tf-idf example: Inc.ltc

**Document:** *car insurance auto insurance*

**Query:** *best car insurance*

---

<table>
<thead>
<tr>
<th>Term</th>
<th>tf-raw</th>
<th>tf-wt</th>
<th>df</th>
<th>idf</th>
<th>wt</th>
<th>n’lize</th>
<th>tf-raw</th>
<th>tf-wt</th>
<th>wt</th>
<th>n’lize</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto</td>
<td>0</td>
<td>0</td>
<td>5000</td>
<td>2.3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.52</td>
</tr>
<tr>
<td>best</td>
<td>1</td>
<td>1</td>
<td>50000</td>
<td>1.3</td>
<td>1.3</td>
<td>0.34</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>car</td>
<td>1</td>
<td>1</td>
<td>10000</td>
<td>2.0</td>
<td>2.0</td>
<td>0.52</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.52</td>
</tr>
<tr>
<td>insurance</td>
<td>1</td>
<td>1</td>
<td>1000</td>
<td>3.0</td>
<td>3.0</td>
<td>0.78</td>
<td>2</td>
<td>1.3</td>
<td>1.3</td>
<td>0.68</td>
</tr>
</tbody>
</table>

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

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

Score $= 0 + 0 + 0.27 + 0.53 = 0.8$
### Binary → count → weight matrix

<table>
<thead>
<tr>
<th></th>
<th>Antony and Cleopatra</th>
<th>Julius Caesar</th>
<th>The Tempest</th>
<th>Hamlet</th>
<th>Othello</th>
<th>Macbeth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Antony</td>
<td>5.25</td>
<td>3.18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.35</td>
</tr>
<tr>
<td>Brutus</td>
<td>1.21</td>
<td>6.1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Caesar</td>
<td>8.59</td>
<td>2.54</td>
<td>0</td>
<td>1.51</td>
<td>0.25</td>
<td>0</td>
</tr>
<tr>
<td>Calpurnia</td>
<td>0</td>
<td>1.54</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cleopatra</td>
<td>2.85</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>mercy</td>
<td>1.51</td>
<td>0</td>
<td>1.9</td>
<td>0.12</td>
<td>5.25</td>
<td>0.88</td>
</tr>
<tr>
<td>worser</td>
<td>1.37</td>
<td>0</td>
<td>0.11</td>
<td>4.15</td>
<td>0.25</td>
<td>1.95</td>
</tr>
</tbody>
</table>

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

- So we have a $|V|$-dimensional vector space
- Terms are axes of the space
- Documents are points or vectors in this space
- Very high-dimensional: tens of millions of dimensions when you apply this to a web search engine
- These are very sparse vectors - most entries are zero.
Queries as vectors

- **Key idea 1:** Do the same for queries: represent them as vectors in the space
- **Key idea 2:** Rank documents according to their proximity to the query in this space
  - proximity = similarity of vectors
  - proximity \(\approx\) inverse of distance
  - We do this because we want to get away from the you’re-either-in-or-out Boolean model.
  - Instead: rank more relevant documents higher than less relevant documents
Euclidean 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.
cosine(query, document)

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

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

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

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

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

for \(q, d\) length-normalized.
How similar are these 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 contd.

- Log frequency weighting
- 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>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>

<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

\texttt{CosineScore}(q)

1. float Scores\[N\] = 0
2. float Length\[N\]
3. for each query term \(t\)
4. do calculate \(w_{t,q}\) and fetch postings list for \(t\)
5. for each pair\((d, tf_{t,d})\) in postings list
6. do Scores\[d\] += \(w_{t,d} \times w_{t,q}\)
7. Read the array Length
8. for each \(d\)
9. do Scores\[d\] = Scores\[d\]/Length\[d\]
10. return Top \(K\) components of Scores[]
Summary – vector space models

- 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
Ranked retrieval

LANGUAGE MODELS
Trouble with frequency-based models

- Too literal
- Can’t deal with misspellings, synonyms etc.
- Natural language queries are hard to deal with if you don’t address these difficulties
Language Model

• *Unigram language model*
  – probability distribution over the words in a language
  – generation of text consists of pulling words out of a “bucket” according to the probability distribution and replacing them

• *N-gram language model*
  – some applications use bigram and trigram language models where probabilities depend on previous words
Semantic distance

Measuring psychophysical distances between perceptual stimuli is hard.

Words that co-occur in many situations correspond to psychophysically close stimuli.

NGD = \[ \log \left( \frac{\text{Webpages with } X}{\text{All webpages indexed by Google}} \right) - \log \left( \frac{\text{Webpages with both}}{\text{Webpages with } Y} \right) \]

Words correspond to stimuli, webpages are sets of words co-occurring to describe situations the documents are about.

cat, dog, pet, home, cute
cat, tiger, lion, puma, jaguar
zoo, bear, wolf, lion, tiger

measurable distances indirectly using semantic information.
Sample topic
A topic in a document or query can be represented as a language model

- i.e., words that tend to occur often when discussing a topic will have high probabilities in the corresponding language model
- The basic assumption is that words cluster in semantic space

Multinomial distribution over words

- text is modeled as a finite sequence of words, where there are \( t \) possible words at each point in the sequence
- commonly used, but not only possibility
- doesn’t model burstiness
Has interesting applications

Completely Contextual
- Mango, Apple, Banana

Diversity
- Mango, Apple, Banana
- Mango, Orangutans

Anomalousness
- Mango, Apple, Banana
- Treadmill

Health conscious monkeys and orangutans prefer fruits for lunch after working out on the treadmill

Diversity + Anomalousness = Conceptual Incongruity
LMs for Retrieval

• 3 possibilities:
  – probability of generating the query text from a document language model
  – probability of generating the document text from a query language model
  – comparing the language models representing the query and document topics

• Models of topical relevance
Query-Likelihood Model

• Rank documents by the probability that the query could be generated by the document model (i.e. same topic)
• Given query, start with $P(D|Q)$
• Using Bayes’ Rule

$$p(D|Q) \overset{rank}{=} P(Q|D)P(D)$$

• Assuming prior is uniform, unigram model

$$P(Q|D) = \prod_{i=1}^{n} P(q_i|D)$$
Other query constructions

\[ \text{bel}_{not}(q) = 1 - p_1 \]

\[ \text{bel}_{or}(q) = 1 - \prod_{i}^{n} (1 - p_i) \]

\[ \text{bel}_{and}(q) = \prod_{i}^{n} p_i \]

\[ \text{bel}_{wand}(q) = \prod_{i}^{n} p_i^{wt_i} \]

\[ \text{bel}_{max}(q) = \max\{p_1, p_2, \ldots, p_n\} \]

\[ \text{bel}_{sum}(q) = \frac{\sum_{i}^{n} p_i}{n} \]

\[ \text{bel}_{wsum}(q) = \frac{\sum_{i}^{n} wt_i p_i}{\sum_{i}^{n} wt_i} \]
Estimating Probabilities

• Obvious estimate for unigram probabilities is

\[ P(q_i | D) = \frac{f_{q_i,D}}{|D|} \]

• Maximum likelihood estimate
  – makes the observed value of \( f_{q,D} \) most likely

• If query words are missing from document, score will be zero
  – Missing 1 out of 4 query words same as missing 3 out of 4
Smoothing

• Document texts are a *sample* from the language model
  – Missing words should not have zero probability of occurring

• *Smoothing* is a technique for estimating probabilities for missing (or unseen) words
  – lower (or *discount*) the probability estimates for words that are seen in the document text
  – assign that “left-over” probability to the estimates for the words that are not seen in the text
Estimating Probabilities

• Estimate for unseen words is \( \alpha_D P(q_i | C) \)
  
  – \( P(q_i | C) \) is the probability for query word \( i \) in the collection language model for collection \( C \) (background probability)
  
  – \( \alpha_D \) is a parameter

• Estimate for words that occur is
  
  \[
  (1 - \alpha_D) P(q_i | D) + \alpha_D P(q_i | C)
  \]

• Different forms of estimation come from different \( \alpha_D \)
Jelinek-Mercer Smoothing

• $\alpha_D$ is a constant, $\lambda$
• Gives estimate of

$$p(q_i | D) = (1 - \lambda) \frac{f_{q_i, D}}{|D|} + \lambda \frac{c_{q_i}}{|C|}$$

• Ranking score

$$P(Q | D) = \prod_{i=1}^{n} ((1 - \lambda) \frac{f_{q_i, D}}{|D|} + \lambda \frac{c_{q_i}}{|C|})$$

• Use logs for convenience
  – accuracy problems multiplying small numbers

$$\log P(Q | D) = \sum_{i=1}^{n} \log\left( (1 - \lambda) \frac{f_{q_i, D}}{|D|} + \lambda \frac{c_{q_i}}{|C|} \right)$$
Compare with \( tf.idf \)

\[
\log P(Q|D) = \sum_{i=1}^{n} \log((1 - \lambda) \frac{f_{q_i,D}}{|D|} + \lambda \frac{c_{q_i}}{|C|})
\]

\[
= \sum_{i: f_{q_i,D} > 0} \log((1 - \lambda) \frac{f_{q_i,D}}{|D|} + \lambda \frac{c_{q_i}}{|C|}) + \sum_{i: f_{q_i,D} = 0} \log(\lambda \frac{c_{q_i}}{|C|})
\]

\[
= \sum_{i: f_{q_i,D} > 0} \log \left( \frac{(1 - \lambda) \frac{f_{q_i,D}}{|D|} + \lambda \frac{c_{q_i}}{|C|}}{\lambda \frac{c_{q_i}}{|C|}} \right) + \sum_{i=1}^{n} \log(\lambda \frac{c_{q_i}}{|C|})
\]

\[
\text{rank} = \sum_{i: f_{q_i,D} > 0} \log \left( \frac{(1 - \lambda) \frac{f_{q_i,D}}{|D|} + 1}{\lambda \frac{c_{q_i}}{|C|}} \right)
\]

- proportional to the term frequency, inversely proportional to the collection frequency
Dirichlet Smoothing

• $\alpha_D$ depends on document length

$$\alpha_D = \frac{\mu}{|D|+\mu}$$

• Gives probability estimation of

$$p(q_i | D) = \frac{f_{q_i,D} + \mu \frac{c_{q_i}}{|C|}}{|D|+\mu}$$

• and document score

$$\log P(Q | D) = \sum_{i=1}^{n} \log \frac{f_{q_i,D} + \mu \frac{c_{q_i}}{|C|}}{|D|+\mu}$$
Query Likelihood Example

• For the term “president”
  \[ f_{q_i,D} = 15, \ c_{q_i} = 160,000 \]
• For the term “lincoln”
  \[ f_{q_i,D} = 25, \ c_{q_i} = 2,400 \]
• document \(|d|\) is assumed to be 1,800 words long
• collection is assumed to be \(10^9\) words long
  \[ – \ 500,000 \text{ documents times an average of 2,000 words} \]
• \(\mu = 2,000\)
Query Likelihood Example

\[ QL(Q, D) = \log \frac{15 + 2000 \times (1.6 \times 10^5 / 10^9)}{1800 + 2000} \]
\[ + \log \frac{25 + 2000 \times (2400 / 10^9)}{1800 + 2000} \]
\[ = \log(15.32/3800) + \log(25.005/3800) \]
\[ = -5.51 + -5.02 = -10.53 \]

- Negative number because summing logs of small numbers
## Query Likelihood Example

<table>
<thead>
<tr>
<th>Frequency of “president”</th>
<th>Frequency of “lincoln”</th>
<th>QL score</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>25</td>
<td>-10.53</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>-13.75</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>-19.05</td>
</tr>
<tr>
<td>1</td>
<td>25</td>
<td>-12.99</td>
</tr>
<tr>
<td>0</td>
<td>25</td>
<td>-14.40</td>
</tr>
</tbody>
</table>
Going beyond tf.idf

\[ p(q|D) = \frac{f_{q,D}}{|D|} \]

\[ p(q|D) = \sum_M p(q|M)p(M|D) \]

\[ p(q|M) = \frac{f_{q,M}}{|M|} \]

\[ p(D|M) = \prod_w p(w|M) \]