Department of Computer Science
The University of Melbourne
COMP90042 WEB SEARCH AND TEXT ANALYSIS (Semester 1, 2017)

Workshop exercises: Week 9

Discussion

1. Why is evaluating an Information Retrieval engine difficult?

   - Evaluation is generally in terms of **relevance**, but it's very difficult to characterise that in a precise manner.
   - The collection of documents is usually too large to manually assess every document in the collection for relevance.

   (a) What are some assumptions that we need to make, and why must we make them?

   - As the lectures state:
     - Retrieval is *ad hoc*, so that we don't need to take into account the user and their behaviour when assessing the relevance of a particular document (which would be nice, but impractical).
     - Relevance is binary (1/0), although you can find some metrics that can account for partially–relevant results. Binary relevance judgements easier to perform, however (and scoring the partial relevance is notoriously difficult, even for humans).
     - Relevance judgements are independent — so that we can label individual documents in an off–line manner, rather than the results for a query as a whole.

   (b) Why is recall usually impossible to calculate? Why do we (usually) not care?

   - To calculate recall, we would need to know all of the **false negatives** — that is, the truly relevant documents that weren't returned by the system. The only way of determining this, however, is to manually examine every document in the collection, which is impractical, except for tiny collections.
   - Generally speaking, we don't care about recall, because the user typically doesn't want to see **every** relevant document in the collection (there are some counter–examples to this, most notably in the legal and biomedical domains). Remember that we're defining relevance as **meeting the user's information need** — if it has been met by one document, the user is typically satisfied, and they no longer wish to examine other potentially relevant documents.

2. Recall the Okapi BM25 term weighting formula:

$$w_t = \log \frac{N - f_t + 0.5}{f_t + 0.5} \times \frac{(k_1 + 1) f_{d,t}}{k_1((1-b) + b\frac{L_d}{L_{\text{avg}}}) + f_{d,t}} \times \frac{(k_3 + 1) f_{q,t}}{k_3 + f_{q,t}}$$

   (a) What are its parameters, and what do they signify?

- k_1 controls the weight of the term frequencies: high values mean that the weight for a term in the query is mostly linear in the frequency of the term in a document; low values mean that the term frequencies are mostly ignored, and only their presence/absence in the documents matters.
- k_2 isn't in this model. (!?)
- k_3 controls the term frequency within the query — so that, where terms are repeated in the query, they contribute more to the document ranking. (Note that this is often set to 0, so that query terms are only binary.)
- b controls the penalty for long documents: when b is close to 0, document length is mostly excluded from the model; when b is large, short documents are strongly preferred to long documents (all else equal).

(b) Using suitable default parameter settings, find the ranking for the query `apple ibm`, using the BM25 term–weighting model, as calculated over the following collection:

	`apple`	`ibm`	`lemon`	`sun`
D_1	4	0	1	1
D_2	5	0	5	0
D_3	2	5	0	0
D_4	1	2	1	7
D_5	1	1	3	0

- Let's use $k_1 = 1.5$, $b = 0.5$, $k_3 = 0$ (although there are no repeated terms in the query, so k_3 doesn't really matter here).
- First, we're going to need some basic statistics about the lengths of the documents. Unlike a typical TF-IDF model, the length of the document in Okapi BM25 is just a count of terms. So, the length of document 1 is $4 + 0 + 1 + 1 = 6$ ($L_1 = 6$), the length of document 2 is 10, and so on.
- The average length of documents in this collection can be found by summing all of the terms, and dividing through by the number of documents; in this case, that will be $\frac{39}{5} = 7.8$ ($L_{\text{avg}} = 7.8$).
- Now, let's consider `apple`, which occurs in all 5 documents ($f_a = 5$; $N = 5$):
  - `apple` occurs in D_1 4 times ($f_{1,a} = 4$), so its BM25 weight becomes:

$$\begin{aligned}
w_{1,a} &= \log\frac{N - f_a + 0.5}{f_a + 0.5} \times \frac{(k_1 + 1)f_{1,a}}{k_1((1-b) + b\frac{L_1}{L_{\text{avg}}}) + f_{1,a}} \times \frac{(k_3 + 1)f_{q,a}}{k_3 + f_{q,a}} \\
&= \log\frac{5 - 5 + 0.5}{5 + 0.5} \times \frac{(1.5 + 1)(4)}{1.5((1 - 0.5) + 0.5(\frac{6}{7.8}) + 4} \times \frac{(0 + 1)(1)}{0 + 1} \\
&\approx \log\frac{0.5}{5.5} \times \frac{10}{5.33} \times \frac{1}{1} \approx -1.95
\end{aligned}$$

  - We can observe a couple of important things here:
    - For $k_3 = 0$, the third term will always be 1, and it can be ignored from the calculation. (Where query terms are not repeated, it will be a constant.)
    - Term weights in the BM25 model can be negative, if the term appears in greater than half of the documents in the collection. This

is unlikely to be a problem in many non-trivial collections, as such a common word is a stop–word, and we would simply exclude it from the collection.

- `apple` occurs in D_2 5 times ($f_{2,a} = 5$), so its BM25 weight becomes:

$$\begin{aligned} w_{2,a} &= \log \frac{N - f_a + 0.5}{f_a + 0.5} \times \frac{(k_1+1)f_{2,a}}{k_1((1-b)+b\frac{L_1}{L_{\text{avg}}}) + f_{2,a}} \\ &= \log \frac{5-5+0.5}{5+0.5} \times \frac{(1.5+1)(5)}{1.5((1-0.5)+0.5(\frac{10}{7.8}) + 5} \\ &\approx \log \frac{0.5}{5.5} \times \frac{12.5}{6.71} \approx -1.94 \end{aligned}$$

- We can see that this term has almost the same weight: it has a higher term frequency, but in a document longer than average.
- For the other three documents:

$$\begin{aligned} w_{3,a} &= \log \frac{5-5+0.5}{5+0.5} \times \frac{(1.5+1)(2)}{1.5((1-0.5)+0.5(\frac{7}{7.8}) + 2} \\ &\approx \log \frac{0.5}{5.5} \times \frac{5}{3.42} \approx -1.52 \\ w_{4,a} &= \log \frac{5-5+0.5}{5+0.5} \times \frac{(1.5+1)(1)}{1.5((1-0.5)+0.5(\frac{11}{7.8}) + 1} \\ &\approx \log \frac{0.5}{5.5} \times \frac{2.5}{2.81} \approx -0.93 \\ w_{5,a} &= \log \frac{5-5+0.5}{5+0.5} \times \frac{(1.5+1)(1)}{1.5((1-0.5)+0.5(\frac{5}{7.8}) + 1} \\ &\approx \log \frac{0.5}{5.5} \times \frac{2.5}{2.23} \approx -1.17 \end{aligned}$$

• `ibm` occurs in only 3 documents ($f_i = 3$), but that's still more than half of the collection:

- `ibm` isn't present in D_1 or D_2, so the weight will be 0. (We can solve this explicitly, but it's enough to simply observe that $(k_1+1)f_{1,i}$ is zero, and we're taking a product.)
- For the other three documents:

$$\begin{aligned} w_{3,i} &= \log \frac{5-3+0.5}{3+0.5} \times \frac{(1.5+1)(5)}{1.5((1-0.5)+0.5(\frac{7}{7.8}) + 5} \\ &\approx \log \frac{2.5}{3.5} \times \frac{12.5}{6.42} \approx -0.28 \\ w_{4,i} &= \log \frac{5-3+0.5}{3+0.5} \times \frac{(1.5+1)(2)}{1.5((1-0.5)+0.5(\frac{11}{7.8}) + 2} \\ &\approx \log \frac{2.5}{3.5} \times \frac{5}{3.81} \approx -0.19 \\ w_{5,i} &= \log \frac{5-3+0.5}{3+0.5} \times \frac{(1.5+1)(1)}{1.5((1-0.5)+0.5(\frac{5}{7.8}) + 1} \\ &\approx \log \frac{2.5}{3.5} \times \frac{2.5}{2.23} \approx -0.16 \end{aligned}$$

- That process might have felt slightly exhausting, but finding the final scores for each document is quite easy by comparison: we simply sum the BM25 values for the terms in the query:

$$\begin{aligned} D_1 &: -1.95 + 0 = -1.95 \\ D_2 &: -1.94 + 0 = -1.94 \\ D_3 &: -1.52 + -0.28 = -1.80 \\ D_4 &: -0.93 + -0.19 = -1.12 \\ D_5 &: -1.17 + -0.16 = -1.33 \end{aligned}$$

- In a somewhat realistic scenario (which this isn't), we would probably not return any of these documents. But the "best" here is the one with the greatest score: D_4 (mostly because the presence of these undesirable terms is off-set by the length of the document!).
- You might think that this is a good demonstration that BM25 shouldn't work, but in fact it does, mostly because the assumptions that go into the model (in particular, the IDF assumptions), only make sense when applied to real document collections, and not toy collections like this one.

3. How can we use a **language model** to solve the above problem? Is this a sensible strategy? Why or why not?

   - We can construct a smoothed uni–gram language model for each of the documents in our collection. We can then find the document whose model gives the greatest probability to the "sentence" defined by the query terms.

   (a) What is the logic behind using a **unigram** model here? When might higher–order models be preferable, but why do we not usually use them?

   - Basically, that the order of query terms is unimportant. If we know that we have a phrasal query, we might be better off using a higher–order model, to account for the query terms being present in order.
   - Although it has been observed that a surprisingly high proportion of non–phrasal queries evaluate successfully when treated as phrasal (based on the Excite logs, more than 40%!) — there are too many queries where the context is irrelevant or misleading to justify using more than a uni–gram model.

   (b) Observe that the "Dirichlet smoothed" LM from the lecture slides is the same as the so–called "Jelinek–Mercer smoothed" language model from KT:

$$P(d|q) = \prod_{t \in q} \left(\frac{|d|}{|d| + \mu} \times \frac{f_{d,t}}{|d|} + \frac{\mu}{|d| + \mu} \times \frac{F_t}{F} \right)$$

   - The derivation is very simple, but it is left as an exercise to the reader.