School of Computing and Information Systems
The University of Melbourne
COMP90042
WEB SEARCH AND TEXT ANALYSIS (Semester 1, 2019)

Sample solutions for discussion exercises: Week 2

## Discussion

1. Give some examples of text processing applications that you use on a daily basis.
   - There are lots! For example, Google (or other web search engines), Siri (or other speech-to-text systems), predictive messaging, spelling correction, machine translation, and so on.
2. What is **tokenisation** and why is it important?
   - Tokenisation is the act of transforming a (long) document into a set of meaningful substrings, so that we can compare with other (long) documents.
   - In general, a document is too long — and contains too much information — to manipulate directly. There are some counter-examples, like **language identification**, which we need to perform before we decide how to tokenise anyway.

   (a) What are **stemming** and **lemmatisation**, and how are they different? Give examples from the `WSTA_N1_preprocessing` iPython notebook.
   - Both stemming and lemmatisation are mechanisms for transforming a token into a canonical (base, normalised) form. For example, turning the token *walking* into its base form *walk*.
   - Both operate by applying a series of rewrite operations to remove or replace (parts of) affixes (primarily suffixes). (In English, anyway.)
   - However, lemmatisation works in conjunction with a **lexicon**: a list of valid words in the language. The goal is to turn the input token into an element of this list (a valid word) using the rewrite rules. If the re-write rules can't be used to transform the token into a valid word, then the token is left alone. (For example, the token *lemming* wouldn't be transformed into *lemm* because the latter isn't in the word list.)
   - Stemming simply applies the rewrite rules, even if the output is a garbage token (like *lemm*).
   - One further idea is the difference between **inflectional morphology** and **derivational morphology**:
     - Inflectional morphology is the systematic process (in many but not all languages) by which tokens are altered to conform to certain grammatical constraints: for example, if the English noun *teacher* is plural, then it must be represented as *teachers*. The idea is that these changes don't really alter the meaning of the term. Consequently, both stemming and lemmatisation attempt to remove this kind of morphology.

- Derivational morphology is the (semi-)systematic process by which we transform terms of one **class** into a different class (more on this next week). For example, if we would like to make the English verb *teach* into a noun (someone who performs the action of *teaching*), then it must be represented as *teacher*. This kind of morphology tends to produce terms that differ (perhaps subtly) in meaning, and the two separate forms are usually **both** listed in the lexicon. Consequently, lemmatisation doesn't usually remove derivational morphology in its normalisation process, but stemming usually does.

- Another example, from the notebook, is the token *this*. Using the lemmatiser, the token remains unchanged, because it is already listed in the lexicon. The stemmer, however, strips the *-s* suffix, so that we end up with *thi*.

3. Compare using a **term–document matrix** vs. an **inverted index** for resolving a ranked query efficiently.

   - Assuming Term-at-a-Time processing (not WAND) some kind of TF–IDF vector space model using cosine similarity, using **accumulators** to keep track of each document's score:
   - For the term–document matrix, we will need to read the TDM value of **every** document for **each** query term. (Note that shortcuts — like skipping documents with a term weight of 0 — save little time here.)
   - For the inverted index, once more we only need to examine each entry in each postings list once; except that, here we incrementing the accumulator weights rather than comparing documents identifiers.
   - Algorithms such as WAND can further improve processing of the inverted index lists.

4. Using the TF-IDF vector space model, using raw term frequency $tf_{d,t}$, $\log \frac{N}{df_t}$ as the inverse document frequency formulation, find the ranking for the query `apple ibm`, based on calculated over the following collection. You should use cosine similarity, but for this question, we will skip the document normalisation step (for time reasons.)

| | `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 | 0 | 1 | 7 |
| $D_5$ | 0 | 1 | 3 | 0 |

   - The similarity metric is $S_{\text{TF-IDF}}(d, Q) = \sum_{t \in Q} tf_{d,t} \times \log \frac{N}{df_t}$ which has to be computed for the query and each document. Where $tf_{d,t}$ is the number of times term $t$ occurs in document $d$ and $df_t$ is the number of documents term $t$ occurs in the collection. $N$ is the total number of documents in the collection..
   - Now, let's consider `apple`, which occurs in 4 documents ($df_a = 4; N = 5$):

- `apple` occurs in $D_1$ 4 times ($tf_{1,a} = 4$), so its TF-IDF weight becomes:

$$w_{1,a} = tf_{1,a} \times \log \frac{N}{df_a} = 4 \times \log \frac{5}{4} = 0.89$$

- Performing the same computation for all documents yields with term `apple`:

$$\begin{aligned} w_{2,a} &= tf_{2,a} \times \log \frac{N}{df_a} = 5 \times \log \frac{5}{4} = 1.12 \\ w_{3,a} &= tf_{3,a} \times \log \frac{N}{df_a} = 2 \times \log \frac{5}{4} = 0.45 \\ w_{4,a} &= tf_{4,a} \times \log \frac{N}{df_a} = 1 \times \log \frac{5}{4} = 0.22 \\ w_{5,a} &= tf_{5,a} \times \log \frac{N}{df_a} = 0 \times \log \frac{5}{4} = 0 \end{aligned}$$

- And then the other terms (`ibm, lemon, sun`). First we'll calculate the idf terms:

| | `apple` | `ibm` | `lemon` | `sun` |
|---|---|---|---|---|
| idf | $\log \frac{5}{4} = 0.22$ | $\log \frac{5}{2} = 0.92$ | $\log \frac{5}{4} = 0.22$ | $\log \frac{5}{2} = 0.92$ |

And use these to produce the remaining TFxIDF scores:

| | `apple` | `ibm` | `lemon` | `sun` |
|---|---|---|---|---|
| $D_1$ | 0.89 | 0 | 0.22 | 0.92 |
| $D_2$ | 1.12 | 0 | 1.12 | 0 |
| $D_3$ | 0.45 | 4.58 | 0 | 0 |
| $D_4$ | 0.22 | 0 | 0.22 | 6.41 |
| $D_5$ | 0 | 0.92 | 0.67 | 0 |

- The IDF component controls the "importance" of each term in the query. The term `apple` occurs in most documents so it contributes less towards identifying relevant documents. Imagine a large text collection of English documents. Most documents will contain multiple occurrences the term "the". So, the IDF adjusts the importance of terms to the overall similarity score of documents.
- On the other hand, consider the term `ibm`, which occurs in only a few documents. Thus, the magnitude of the $w_{,i}$ weights is much higher. The property that less frequent documents are generally more important is also utilised the WAND algorithm (see below) which exploits this property to avoid scoring documents which only contain terms that occur frequently.
- finding the final scores for each document is quite easy by comparison: we simply sum the TF-IDF values for the terms in the query:

$$\begin{aligned} D_1 &: w_{1,a} + w_{1,i} = 0.89 + 0 = 0.89 \\ D_2 &: w_{2,a} + w_{2,i} = 1.12 + 0 = 1.12 \\ D_3 &: w_{3,a} + w_{3,i} = 0.45 + 4.58 = 5.03 \\ D_4 &: w_{4,a} + w_{4,i} = 0.22 + 0 = 0.22 \\ D_5 &: w_{5,a} + w_{5,i} = 0 + 0.92 = 0.92 \end{aligned}$$

- The most "relevant" document according to our TF-IDF metric is $D_3$ by a large margin because it contains a rare term ($df_i = 2$), multiple times!
- You were instructed to skip the document normalisation step. This corresponds to normalising each row vector (document) by its vector magnitude (sum of squares). I.e., The normalisation factor $W_d = \sqrt{\sum_t \left(tf_{d,t} \times \log \frac{N}{df_t}\right)^2}$.

  For completeness, here's how you would do it:

| | `apple` | `ibm` | `lemon` | `sun` | magnitude |
|---|---|---|---|---|---|
| $D_1$ | 0.89 | 0 | 0.22 | 0.92 | 1.30 |
| $D_2$ | 1.12 | 0 | 1.12 | 0 | 1.58 |
| $D_3$ | 0.45 | 4.58 | 0 | 0 | 4.6 |
| $D_4$ | 0.22 | 0 | 0.22 | 6.41 | 6.43 |
| $D_5$ | 0 | 0.92 | 0.67 | 0 | 1.14 |

  And then divide through each row:

| | `apple` | `ibm` | `lemon` | `sun` |
|---|---|---|---|---|
| $D_1$ | 0.69 | 0 | 0.17 | 0.71 |
| $D_2$ | 0.71 | 0 | 0.70 | 0 |
| $D_3$ | 0.10 | 0.99 | 0 | 0 |
| $D_4$ | 0.03 | 0 | 0.03 | 0.999 |
| $D_5$ | 0 | 0.81 | 0.59 | 0 |

  For our query of `apple` and `ibm`, the top scoring document is still $D_3$ with a score of $0.1 + 0.99 = 1.09$.

5. Recall the Okapi BM25 term weighting formula:

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

where $f_t$ is the document frequency of term $t$, $f_{d,t}$ is the term frequency of term $t$ in document $d$ and $f_{q,t}$ is the term frequency of term $t$ in query $q$. $k_1$, $k_3$ and $b$ are parameters with $0 \leq k_1 < \infty$, $0 \leq k_3 < \infty$ and $0 \leq b \leq 1$. $L_d$ is the length of document $d$ and $L_{\text{avg}}$ is the average document length in the collection.

What are its parameters, and what do they signify? How do the components relate to TF (term frequency) and inverse document frequency (IDF)?

(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. In fact, in practice, $k_2 = k_3 = 0$ and $k_1 = 0.9$, $b = 0.4$ gives good performance. These values were determined empirically and will be need to be "tuned" for different text collections.
- $k_3$ 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).
- Note that there are many different similarity metrics out there. In practice we are mainly interested in similarity metrics that can be computed efficiently while giving a reasonable relevance ranking as the results will often be *reranked* (scored again) by a more complex neural model containing many more features such as PageRank etc.

(b) How do the components relate to TF (term frequency) and inverse document frequency (IDF)?

- As discussed above, $k_1$ controls the importance of the term frequencies (TF).
- Term weights (IDF) 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.