School of Computing and Information Systems
The University of Melbourne
COMP90049 Knowledge Technologies (Semester 2, 2019)
Sample Solutions: Week 6

1. What are the four tasks into which a Web-scale information retrieval engine is usually divided? Briefly summarise each one.

   - **Crawling**: finding and downloading as many documents as we can from the web (hopefully all of them, although this isn't possible in practice)
   - **Parsing**: turning each document into a list of tokens (or terms), probably by removing page metadata, case folding, stemming, etc.
   - **Indexing**: building an inverted index out of all of the tokens in our downloaded document collection. (We stop worrying about the original documents at this point.)
   - **Querying**: after the previous three steps have been completed (offline), we are ready to accept user queries (online), in the form of keywords, that we tokenise (in a similar manner to our document collection) and then apply our querying model (e.g. TF-IDF) based on the information in the inverted index, to come up with a document ranking
   - (Optionally) Add-ons: Extras to change the above ranking, based on ad-hoc application of certain factors, for example, PageRank, HITS, click-through data, zones, anchor text, etc

2. When parsing Web pages:

   (a) What is "tokenisation"? What are some common problems that arise in tokenisation of English text? What about other languages?

   - Tokenisation is the act of turning the raw text of a document into tokens that we will use to compare against the (similarly tokenised) query, for example, by removing meta-data and punctuation, folding case, canonicalising for dialect differences, and so on.
   - In English, this might correspond to our notion of "words," although this is less well defined for languages with different approaches to morphology and syntax (many Aboriginal North American languages), and those that don't use whitespace in the same way (Mandarin, for example).
   - Even in English, there are many issues: for example, contractions - is `can't` one token or two? The lectures discuss many examples of this.
   - In Japanese, tokenisation was so bad for such a long time, that keyword search as we know it today was mostly useless, and up until recently, topic based approaches were more popular.

   (b) When we tokenise text, we often canonicalise it. What are these generally accepted as referring to? (Note the terminology is not used consistently in the literature; for example "tokenisation" occasionally refers to both ideas.)

   - Tokenisation, here, means decomposing the larger document into smaller information-bearing units ("tokens") than we can compare against (the keywords present in) our query.
   - Canonicalisation, here, means having a single representation of (each token within) a text, which we can use to sensibly compare one text with another (in our case, a document on the Web, and a query).

   (c) What are some issues that arise when canonicalising text written in English?

   There are various point suggested in the lectures, including:
   - Stopwords
   - Stemming
   - Date/number formatting
   - Dialect variation

- Spelling errors
- Hyphenated tokens
- Compound words
- The genitive 's

Note that English is easy compared to many languages!

3. Zoning is the process of adjusting the weights of tokens, depending on document metadata (for example, whether the token appear in the page title, or an image caption, it might be assigned a different weight). Why is this a stage in the parsing step of an IR engine, and not the indexing or querying step?

- In short, because the document metadata is stripped during the parsing step - we are left with a stream of tokens, and generally no indication about the information necessary to apply zoning (like where in the document each term appeared).
- It's true that the actual weights are stored in the inverted index and (probably) calculated for the TF-IDF model at query time. However, parsing is about building a representation of each document in the collection; to account for the document metadata, we require an alternative parsing strategy.

4. Assume that we have crawled the following "documents":

1) `The South Australian Tourism Commission has defended a marketing strategy which pays celebrities to promote Kangaroo Island tourism to their followers on Twitter.`
2) `Mr O'Loughlin welcomed the attention the use of Twitter had now attracted.`
3) `Some of the tweeting refers to a current television advertisement promoting Kangaroo Island.`
4) `Those used by the Commission have included chef Matt Moran, TV performer Sophie Falkiner and singer Shannon Noll.`
5) `He said there was nothing secretive about the payments to celebrities to tweet the virtues of a tourism destination.`
6) `Marketing director of SA Tourism, David O'Loughlin, said there was no ethical problem with using such marketing and it might continue to be used.`
7) `Depending on their following, celebrities can be paid up to $750 for one tweet about the island.`

- Parse each document into terms.

  Let's consider a term to be a token with whitespace or punctuation etc. (`\b`) on either side. Let's also strip punctuation and fold case, and apply stemming (so that Australian and australia are instances of the same token). We might also remove stop words, although were asked to index some of them below.

- Construct an inverted index over the documents, for (at least) the terms `and, australia, celebrity, commission, island, on, the, to, tweet, twitter`

  Assuming we've stemmed the tokens in the document collection, an inverted index for these terms (with term frequencies in parentheses) might look like the following:

`and`	→ 4(1) → 6(1)
`australia`	→ 1(1)
`celebrity`	→ 1(1) → 5(1) → 7(1)
`commission`	→ 1(1) → 4(1)
`island`	→ 1(1) → 3(1) → 7(1)
`on`	→ 1(1) → 7(1)
`the`	→ 1(1) → 2(2) → 3(1) → 4(1) → 5(2) → 7(1)
`to`	→ 1(2) → 3(1) → 5(2) → 6(1) → 7(1)
`tweet`	→ 3(1) → 5(1) → 7(1)
`twitter`	→ 1(1) → 2(1)

- Using the vector space model and the cosine measure, rank the documents for the query `commission to island on twitter`

(a) Using the weighting function $w_{d,t} = f_{d,t}$ and $w_{q,t} = \frac{N}{f_t}$

For the query `commission to island on twitter`, we take each term and find its weight in the query. To do this, say for `commission`, we observe that there are 7 documents in the collection, and 2 of them contain `commission`, and so on:

$$w_{q,commision} = \frac{N}{f_{commision}} = \frac{7}{2} = 3.5$$

$$w_{q,island} = \frac{7}{3} \approx 2.33$$

$$w_{q,on} = \frac{7}{2} = 3.5$$

$$w_{q,to} = \frac{7}{5} = 1.4$$

$$w_{q,twitter} = \frac{7}{2} = 3.5$$

All of the other term weights in the query are 0, so we can ignore them (and the weight documents are unimportant for the dot-product). The document weights are just the frequency of the term in each document (which we have recorded in our inverted index above). For example, for `island` our representation would look like:

$$w_{d_1,i} = 1$$

$$w_{d_2,i} = 0$$

$$w_{d_3,i} = 1$$

$$w_{d_4,i} = 0$$

$$w_{d_5,i} = 0$$

$$w_{d_6,i} = 0$$

$$w_{d_7,i} = 1$$

The cosine model gives us:

$$\cos(d_1, q) = \frac{\langle 1,1,1,2,1\rangle . \langle 3.5,2.33,3.5,1.4,3.5\rangle}{|\langle 0,1,1,1,1,1,1,2,0,1\rangle| . |\langle 3.5,2.33,3.5,1.4,3.5\rangle|} \approx \frac{15.63}{\sqrt{11}\sqrt{44.15}} \approx 0.709$$

$$\cos(d_2, q) = \frac{\langle 0,0,0,0,1\rangle . \langle 3.5,2.33,3.5,1.4,3.5\rangle}{|\langle 0,0,0,0,0,0,2,0,0,1\rangle| . |\langle 3.5,2.33,3.5,1.4,3.5\rangle|} \approx \frac{3.5}{\sqrt{5}\sqrt{44.15}} \approx 0.236$$

$$\cos(d_3, q) = \frac{\langle 0,1,0,1,0\rangle . \langle 3.5,2.33,3.5,1.4,3.5\rangle}{|\langle 0,0,0,0,1,0,1,1,1,0\rangle| . |\langle 3.5,2.33,3.5,1.4,3.5\rangle|} \approx \frac{15.63}{\sqrt{4}\sqrt{44.15}} \approx 0.281$$

$$\cos(d_4, q) = \frac{\langle 1,0,0,0,0\rangle . \langle 3.5,2.33,3.5,1.4,3.5\rangle}{|\langle 1,0,0,1,0,0,1,0,0,0\rangle| . |\langle 3.5,2.33,3.5,1.4,3.5\rangle|} \approx \frac{15.63}{\sqrt{3}\sqrt{44.15}} \approx 0.304$$

$$\cos(d_5, q) = \frac{\langle 0,0,0,2,0\rangle . \langle 3.5,2.33,3.5,1.4,3.5\rangle}{|\langle 0,0,1,0,0,0,2,2,1,0\rangle| . |\langle 3.5,2.33,3.5,1.4,3.5\rangle|} \approx \frac{2.8}{\sqrt{10}\sqrt{44.15}} \approx 0.133$$

$$\cos(d_6, q) = \frac{\langle 0,0,0,1,0\rangle . \langle 3.5,2.33,3.5,1.4,3.5\rangle}{|\langle 1,0,0,0,0,0,0,1,0,0\rangle| . |\langle 3.5,2.33,3.5,1.4,3.5\rangle|} \approx \frac{1.4}{\sqrt{2}\sqrt{44.15}} \approx 0.149$$

$$\cos(d_7, q) = \frac{\langle 0,1,1,1,0\rangle . \langle 3.5,2.33,3.5,1.4,3.5\rangle}{|\langle 0,0,1,0,1,1,1,1,1,0\rangle| . |\langle 3.5,2.33,3.5,1.4,3.5\rangle|} \approx \frac{7.233}{\sqrt{6}\sqrt{44.15}} \approx 0.444$$

- All of the documents have non-zero similarity, so they all get returned, and the order is: {1,7,4,3,2,6,5}.

- Notice also that each cosine calculation involves the same division by the length of the query (in this case, $\sqrt{44.15}$) - since all we care about is the order of the documents and not the actual score, we can safely leave this term out (but it isn't the cosine anymore, so I'll call it cos' below).

(b) Using the weighting functions $w_{d,t} = 1 + \log_2 f_{d,t}$ and $w_{q,t} = \log_2(1+\frac{N}{f_t})$

Let's observe first that nothing has changed with the document weights: frequencies of 0 still have a weight of 0; frequencies of 1 are now 1+log2(1) = 1+0 = 1; frequencies of 2 are now 1+log2(2) = 1+1 = 2. (Higher frequency terms would have their weights reduced with this model.) This also means that the document lengths are the same.

We can now calculate the query weights according to the new model:

q: <2.170; 1.737; 2.170; 1.263; 2.170>,

and the cosine calculations look similar:

$$\cos(d_1,q) = \frac{\langle 1,1,1,2,1\rangle.\langle 2.17,1.737,2.17,1.263,2.17\rangle}{|\langle 0,1,1,1,1,1,1,2,0,1\rangle|} \approx \frac{10.773}{\sqrt{11}} \approx 3.248$$
$$\cos(d_2,q) = \frac{\langle 0,0,0,0,1\rangle.\langle 2.17,1.737,2.17,1.263,2.17\rangle}{|\langle 0,0,0,0,0,0,2,0,0,1\rangle|} \approx \frac{3.5}{\sqrt{5}} \approx 0.970$$
$$\cos(d_3,q) = \frac{\langle 0,1,0,1,0\rangle.\langle 2.17,1.737,2.17,1.263,2.17\rangle}{|\langle 0,0,0,0,1,0,1,1,1,0\rangle|} \approx \frac{15.63}{\sqrt{4}} \approx 1.5$$
$$\cos(d_4,q) = \frac{\langle 1,0,0,0,0\rangle.\langle 2.17,1.737,2.17,1.263,2.17\rangle}{|\langle 1,0,0,1,0,0,1,0,0,0\rangle|} \approx \frac{15.63}{\sqrt{3}} \approx 1.253$$
$$\cos(d_5,q) = \frac{\langle 0,0,0,2,0\rangle.\langle 2.17,1.737,2.17,1.263,2.17\rangle}{|\langle 0,0,1,0,0,0,2,2,1,0\rangle|} \approx \frac{2.8}{\sqrt{10}} \approx 0.799$$
$$\cos(d_6,q) = \frac{\langle 0,0,0,1,0\rangle.\langle 2.17,1.737,2.17,1.263,2.17\rangle}{|\langle 1,0,0,0,0,0,0,1,0,0\rangle|} \approx \frac{1.4}{\sqrt{2}} \approx 0.893$$
$$\cos(d_7,q) = \frac{\langle 0,1,1,1,0\rangle.\langle 2.17,1.737,2.17,1.263,2.17\rangle}{|\langle 0,0,1,0,1,1,1,1,1,0\rangle|} \approx \frac{7.233}{\sqrt{6}} \approx 2.111$$

- The document ranking has changed a little bit this time (because the relative differences between common terms and rare terms is smaller due to the log): {1,7,3,4,2,6,5}
- Also note that some of the values are greater than 1, because I left out the division by the length of the query (cos'), which gives a factor of $\sqrt{18.74}$ in the denominator- it doesn't change the order though.

5. For question 3, suppose there is an accumulator limit of 2 and that query terms are processed in order of decreasing rarity — are the same top two documents found? What are the similarities of the top two documents?

We're using a limiting approach toward accumulator usage: we have two accumulators, which I'm going to call A1 and A2 (which will be associated with their corresponding documents $A_{d,1}$ and $A_{d,2}$).

We'll process terms in order of decreasing rarity, which means that the rarest terms are processed first, down to the commonest terms - in our TF-IDF model, this means that the terms with the greatest $w_{q,t}$ value will be the rarest, and so on. (We'll use the model from Question3, part (b)).

The terms commission, on, and twitter are tied for the highest weight (they each occur twice in the collection). In practice, we'd probably do some kind of tie-break - say, in lexical order, or reverse lexical order, whichever allows faster processing, or maybe chosen randomly - in this case, let's consider what happens for each of these terms:

[Approach 1] : If we process `commission` first:

- The inverted list for `commission` contains document 1 and document 4.
- We first look at document 1, where commission occurs once. There's a free accumulator (A_1), so we update it with this document ($A_{d,1} = 1$) and increment the (initially zero) value with the dot-product component for this term:

$$A_1 = A_1 + w_{q,commission} \times w_{d1,commission} = 0 + 2.17 \times 1 = 2.17$$

- Next, we look at document 4, which also has one commission. There's a free accumulator (A_2), so we update it with this document ($A_{d,2} = 4$), and:

$$A_2 = A_2 + w_{q,}commission \times w_{d4,commission} = 0 + 2.17 \times 1 = 2.17$$

- If there were more documents, we'd be out of luck, because we've run out of accumulators. (But some models let us over-write accumulators if the score for this term would out-weigh the lowest accumulator value; for example, if the next document had two commissions, its accumulator value would be 4.340 and that document would replace, say, document 4.)
- We now process one of `on` or `twitter`. In this case, it doesn't matter which order we consider them, so let's do first.
- The inverted list for on contains document 1 and document 7.
- We first look at document 1, with one `on`, for which there is an accumulator (A1). We update its value with the dot-product component for this term:

$$A_1 = A_1 + w_{q,on} \times w_{d1,on} = 2.17 + 2.17 \times 1 = 4.34$$

- Next, we have document 7. There's no accumulator for that document, so we ignore it. (But again, some models will calculate its dot-product and replace the lowest-valued accumulator if this term would be better.)
- The next best term is `twitter`, which has documents 1 and 2 in its inverted list (both with a frequency of 1).
- For document 1 we will have:

$$A_1 = A_1 + w_{q,twitter} \times w_{d1,twitter} = 4.34 + 2.17 \times 1 = 6.51$$

- For document 2, there is no accumulator.
- The next best term is `island`, with documents 1, 3, and 7 in its list (all with frequency 1). We update document 1 again, and ignore the other two, because they don't have accumulators:

$$A_1 = A_1 + w_{q,island} \times w_{d1,island} = 6.51 + 1.737 \times 1 = 8.247$$

- The final term is `to`, with document 1 twice, document 3 once, document 5 twice, and documents 6 and 7 once. Again, document 1 is the only one with an accumulator:

$$A_1 = A_1 + w_{q,to} \times w_{d1,to} = 8.247 + 1.263 \times 2 = 10.773$$

- Note that the document 5 would replace document 4 at this point (1.263 x 2 = 2.526 > 2.170) if our model allows it.
- The final step is to divide through by the length of the documents that correspond to accumulators:

$$A_1 = \frac{A_1}{W_{d1}} = \frac{10.773}{\sqrt{11}} \approx 3.248$$

$$A2 = \frac{A_2}{W_{d4}} = \frac{2.17}{\sqrt{3}} \approx 1.253$$

(Or if we'd replaced 4 with 5: $A_2 = \frac{A_2}{W_{d5}} = \frac{2.526}{\sqrt{10}} \approx 0.799$)

- We'd then return the documents from our accumulators from highest weight to lowest, namely 1 then 4 (or 1 then 5).

[Approach 2] : What if we process `on` first:

- The inverted list for `on` contains document 1 ($A_{d,1}$) and document 7 ($A_{d,2}$):

  $A_1 = A_1 + w_{q,on} \times w_{d1,on} = 0 + 2.17 \times 1 = 2.17$

  $A_2 = A_2 + w_{q,on} \times w_{d7,on} = 0 + 2.17 \times 1 = 2.17$

- The next two terms are `commission` and `twitter`: both of them have document 1 in their inverted list, and other documents that don't have accumulators:

  $A_1 = A_1 + w_{q,commision} \times w_{d1,commision} = 2.17 + 2.17 \times 1 = 4.34$

  $A_1 = A_1 + w_{q,twitter} \times w_{d1,twitter} = 4.34 + 2.17 \times 1 = 6.51$

- `island` has both 1 and 7, as well as 3, for which there is no accumulator:

  $A_1 = A_1 + w_{q,island} \times w_{d1,island} = 6.51 + 1.737 \times 1 = 8.247$

  $A_2 = A_2 + w_{q,island} \times w_{d7,island} = 2.17 + 1.737 \times 1 = 3.907$

- `to` has both 1 (twice) and 7 (once), as well as other documents without accumulators:

  $A_1 = A_1 + w_{q,to} \times w_{d1,to} = 8.247 + 1.263 \times 2 = 10.773$

  $A_2 = A_2 + w_{q,to} \times w_{d7,to} = 3.907 + 1.263 \times 1 = 5.17$

- We normalise (Note that none of the other documents will surpass 5.170 this time.):

  $A_1 = \frac{A_1}{W_{d1}} = \frac{10.773}{\sqrt{11}} \approx 3.248$

  $A2 = \frac{A_2}{W_{d7}} = \frac{5.17}{\sqrt{6}} \approx 2.111$

- This time, we return document 1, followed by document 7.

[Approach 3] : What if we process `twitter` first? Well, the calculations end up looking a lot like those of `commission` above:

- The inverted list for twitter contains document 1 ($A_{d,1}$) and document 2 ($A_{d,2}$):

  $A_1 = A_1 + w_{q,twitter} \times w_{d1,twitter} = 0 + 2.17 \times 1 = 2.17$

  $A_2 = A_2 + w_{q,twitter} \times w_{d2,twitter} = 0 + 2.17 \times 1 = 2.17$

- The next terms are `commission` and `on`: both have document 1, neither has document 2:

  $A_1 = A_1 + w_{q,commision} \times w_{d1,commision} = 2.17 + 2.17 \times 1 = 4.34$

  $A_1 = A_1 + w_{q,on} \times w_{d1,on} = 4.34 + 2.17 \times 1 = 6.51$

- `island` has document 1, as well as other documents without accumulators:

  $A_1 = A_1 + w_{q,island} \times w_{d1,island} = 6.51 + 1.737 \times 1 = 8.247$

- `to` has document 1 (twice), no document 2, and other documents without accumulators:

  $A_1 = A_1 + w_{q,to} \times w_{d1,to} = 8.247 + 1.263 \times 2 = 10.773$

- (Again, document 5 might replace document 2 at this point (2.526 > 2.170).) We normalise as usual:

  $A1 = \frac{A_1}{W_{d1}} = \frac{10.773}{\sqrt{11}} \approx 3.248$

  $A2 = \frac{A_2}{W_{d2}} = \frac{2.17}{\sqrt{5}} \approx 0.97$

  (Or if we'd replaced 4 with 5: $A2 = \frac{A_2}{W_{d5}} = \frac{2.526}{\sqrt{10}} \approx 0.799$*)*

- And this time, we return 1, followed by 2 (or 1 followed by 5).

So what? Well, obviously the return set is different because we can only return 2 documents, as opposed to all of the documents from the approach without accumulators above.

Note also the different ordering; we still find that document 1 is the top document, but whether we find the second best document (7) depends on the order in which we process the terms. In some models, we actually end up returning the worst document (5).

Even if we'd allowed three accumulators, we'd end up with one of the following document orderings: {1,7,4}, {1,7,2}, {1,7,5}, {1,4,2}, {1,4,5}, {1,2,5} - depending on the order that we process terms, whether we allow for replacements, and how we break ties.

We'd need four accumulators to guarantee that document 7 is returned as a result, regardless of the order that we process the terms.

6. When evaluating a query, why should we begin with the rarest terms (those with greatest $w_{q,t}$)? Is this more true for ranked or Boolean querying?

This is far more true for Boolean querying, where we are typically performing set intersections. By starting with the rarest terms, we can typically reduce the comparisons because each intersection guarantees that the resulting set is as small or smaller than the sets under consideration. If we started with common terms, we would do a lot of comparisons for documents that are going to be pruned in for the rare terms anyway.

For ranked querying, it can still be true for certain approximate methods. But conceptually, if we are just filling in accumulators for each document from the inverted index as we process terms, we don't get any benefit from starting with the high-weight terms. (Unless the number of accumulators we're using is really small, in which case we get some weird effects in missing good documents, as shown above.)