COMP6714 ASSIGNMENT 1 SAMPLE SOLUTION

Q1.

(1) omitted.

(2) Bing obviously expand the query to include accents/umlets.

(3) (a) Neuro-linguistic: "Neuro-linguistic"
(b) otta: "otta",
(c) Neuro-linguistic AND otta: "Neuro-linguistic" "otta"
(d) Neuro-linguistic OR otta: "Neuro-linguistic" OR "otta"
(e) Neuro-linguistic /1 otta: "Neuro-linguistic otta" OR "otta Neuro-linguistic"
(f) bugle: "bugle"
(g) bugle bugle: "bugle" "bugle"
(h) bugle bugle bugle: "bugle" "bugle" "bugle"

No. Let (estimated) query results of A and B be $|A|$ and $|B|$, respectively. Then $|A \cap B| \in [0, \min(|A|, |B|)]$. It is not observed from google's estimates. In fact, if we know the entire collection has $|C|$ documents in total, we can improve the lower bound estimate as $\max(0, |A| + |B| - |C|)$.

Q2. (*20 marks*)

Rearrange the lists as follows

```
fools : 2: <1,17,74,222>;       4: <8,78,108,458>;       7: <3,13,23,193>;
rush  : 2: <2,66,194,321,702>;  4: <9,69,149,429,569>;   7: <4,14,404>;
in    : 2: <3,37,76,444,851>;   4: <10,20,110,470,500>;  7: <5,15,25,195>;

angels: 2: <36,174,252,651>;    4: <12,22,102,432>;      7: <17>;
fear  : 2: <87,704,722,901>;    4: <13,43,113,433>;      7: <18,328,528>;
to    : 2: <47,86,234,999>;     4: <14,24,774,944>;      7: <199,319,599,709>;
tread : 2: <57,94,333>;         4: <15,35,155>;          7: <20,320>;
```

(1) (a) "fools rush in": doc=2, pos=1–3; doc=4, pos=8–10, doc=7, pos=3–5, 13–15.
(b) "fools rush in" AND "angels fear to tread": the second condition matches docs 4 (pos=12–15) only. So the final results is doc 4, and two match regions: 8–10 and 12–15.

(2) both `where` and `in` appear in the doc=7, pos=15.

Q3.

(1) The algorithm scans both input lists only once, with the help of a "buffer" l whose max size is k. In the worst case, in each iteration pp_1 moves to the next position,

all items in l are deleted (because they are far from the current pp_1). Hence, the complexity is $O(|p_1| + |p_2| + |p_1| \cdot k)$.[1]

Note that it might be tempting to think about the case when every thing in one list joins with everything in the other list. However, since the positions are all distinct, the maximum positions that can be joined (i.e., within distance of k) is bounded by k. This will give us the same bound as above.

Rather than using a linear search to remove invalid items in l, we can use a binary search as the items in l are in ascending order. However, for binary search to pay off, we need to avoid linearly invalidating those items. Hence, we could use a circular array of size $2k$ and keep two pointers as its head and tail. This brings down the cost to $O(|p_1| + |p_2| + |p_1| \cdot \log k)$.

(2) Assume each document has been divided into a set of paragraph, and each paragraph divided into a set of sentences. We shall record the position of a term in a document in the following scheme:

`paragraphid`|`sentenceid`.`position`

where the `paragraphid` is the paragraph id within the document, and the `sentenceid` is the sentence id within the paragraph, and the `position` is the position (of the term) in the *document* (not in the sentence).

- For `/k` queries, we just extract the `position` and process as usual.
- For `/S` queries, we just extract the `sentenceid` and process as usual.
- For `/P` queries, we just extract the `paragraphid` and process as usual.

There is another possible modification. We include two special tokens, one corresponding to each sentence end position and the other to each paragraph end position. In order to answer, e.g., query Q with `/S` operator, we just perform list intersection of keywords in Q and the inverted list for sentence-end. Some changes are needed so that whenever we encounter another sentence end, we output the current "valid" occurrences of keywords in Q (if any), and then force their cursors to move to the first position beyond the current sentence end. This method is also known as the *extent list approach* (c.f., Chap 5.3.4 in [CMS09]).

Q4.

Immediate merge. Always one sub-index.

No merge. Let $n := \lceil \frac{|C|}{M} \rceil$. No merge will create n sub-indexes.

Logarithmic merge. Notice the 1-to-1 correspondence between the sub-indexes and the binary format of the number of sub-index created.

Therefore, the number of indexes is $f(n)$, where $f(n)$ gives the number of 1 in the binary representation of n. This is known as the *Hamming weight*[2]. Obviously, $f(n) = O(\log \lceil \frac{|C|}{M} \rceil)$.

[1] $|p_i|$ denotes the length of the inverted list p_i.

[2] `https://en.wikipedia.org/wiki/Hamming_weight`

COMP6714 ASSIGNMENT 1

DUE ON 23:59 12 AUG, 2016 (MON)

Q1. (*30 marks*)

Answer the following questions. Note that when we refer to Google, we mean searches via `www.google.com.au`, and when we refer to Bing, we mean the `www.bing.com` search service, but with "Region" choice set to "Only from Australia".

(1) Search "DFA" using Google and Bing. Compare their top-10 results (*not* including the ads). List the web sites returned by both search engines in their top-10 results (you need to include visible screenshots).

(2) Search "ioauen" using Google and Bing. Compare their top-10 results. Describe the possible differences the two search engines have in terms of token normalization, query expansion, and query suggestion.

(3) Translate and write down the following Boolean searches to queries using the (advanced) query syntax provided by Google[1]. Make sure that you disable google's automatic query expansion (e.g., from `otta` to `otto`). You should also record the result numbers estimated by Google.

   (a) Neuro-linguistic
   (b) otta
   (c) Neuro-linguistic AND otta (note: AND means conjunction)
   (d) Neuro-linguistic OR otta (note: OR means disjunction)
   (e) Neuro-linguistic /1 otta (note: /1 means the occurrence of the two terms must be within distance of 1)
   (f) bugle
   (g) bugle bugle
   (h) bugle bugle bugle

   Do the estimated numbers make sense in terms of Boolean logic? What is the upper and lower bounds for the number of query results of the third query based on the number returned from the first and the second queries?

Q2. (*20 marks*)

Shown below is a portion of a positional index in the format:

```
term: doc1: <position1, position2, ...>;
doc2: <position1, position2, ...>;
```

[1] `http://www.google.com.au/support/websearch/bin/answer.py?answer=136861`

```
angels: 2: <36,174,252,651>;   4: <12,22,102,432>;      7: <17>;
fear  : 2: <87,704,722,901>;   4: <13,43,113,433>;      7: <18,328,528>;
fools : 2: <1,17,74,222>;      4: <8,78,108,458>;       7: <3,13,23,193>;
in    : 2: <3,37,76,444,851>;  4: <10,20,110,470,500>;  7: <5,15,25,195>;
rush  : 2: <2,66,194,321,702>; 4: <9,69,149,429,569>;   7: <4,14,404>;
to    : 2: <47,86,234,999>;    4: <14,24,774,944>;      7: <199,319,599,709>;
tread : 2: <57,94,333>;        4: <15,35,155>;          7: <20,320>;
where : 2: <67,124,393,1001>;  4: <11,41,101,421,431>; 7: <15,35,735>;
```

(1) Which document(s) (if any) match each of the following queries where each expression within quotes is a phrase query?
   (a) "fools rush in"
   (b) "fools rush in" AND "angels fear to tread".
   At which positions do the queries match?

(2) There is something wrong with this positional index. What is the problem?

Q3. (*25 marks*)

(1) What is the worst-case time complexity of the algorithm depicted in Figure 2.12 in the MRS08 textbook? Describe a simple modification that improve the time complexity of the algorithm with respect to k.

(2) Some Boolean retrieval systems (e.g., Westlaw) support the following proximity operators: `/k`, `/S` and `/P`. Describe a simple modification to the positional inverted index to support all these three proximity operators.

Q4. (*25 marks*)

How many sub-indexes will the three dynamic indexing methods, namely *immediate merge*, *no merge*, and *logarithmic merge*, create, respectively? Assume that

- We start from scratch.
- We use $|C|$ to denote the total size of the document collection, and M to denote the memory size.
- You can simply assume that by you can create a sub-index of size M after consuming documents of size M.

You need to show your steps.

Submission Instructions

You need to write your solutions to the questions in a pdf file named `ass1.pdf`. You **must**

- include your **name** and **student ID** in the file, and
- the file can be opened correctly on CSE machines.

You need to show the key steps to get the full mark.

Note: Collaboration is allowed. However, each person must independently write up his/her own solution.

You can then submit the file by `give cs6714 ass1 ass1.pdf`.

Late Penalty: **-10%** per day for the first two days, and **-20%** per day for the following days.

COMP6714 (16S2) ASSIGNMENT 2

DUE ON 23:59 31 OCT, 2016 (MON)

Q1 (*25 marks*)

Consider using the maxscore algorithm to find top-2 results for a query with three different terms $\{A, B, C\}$. The scoring function is the BM25 function with $k_1 = k_3 = 2.0$ and $b = 0$.

$$score(d, Q) = \sum_{t \in Q} idf_t \cdot \frac{(k_1+1)tf_{t,d}}{k_1((1-b)+b\frac{L_d}{L_{ave}})+tf_{t,d}} \cdot \frac{(k_3+1)tf_{t,Q}}{k_3+tf_{t,Q}}$$

Answer the following questions. You need to show major steps.
The posting lists are shown below. Each posting consists of document ID and tf.

term	idf	postings
A	6	$(D_1:1),(D_2:8),(D_5:3),(D_8:10)$
B	2	$(D_1:1),(D_5:4),(D_6:1),(D_7:4)$
C	1	$(D_1:1),(D_2:2),(D_4:1),(D_5:2),(D_6:3),(D_8:1),(D_9:1),(D_{10}:3),(D_{11}:7)$

TABLE 1. Posting Lists

(1) Show that the maxscore for each keyword can be computed *without* examining the postings list.
(2) Using the maxscore obtained above, determine the postings that are accessed *for scoring* by the algorithm. You need to assume that each `skipTo(x)` call "magically" moves the cursor **directly** to the first posting with document ID at least x (i.e., it does *not* access any other postings).

Hint 1. *Calculate the maxscore if you know that the maximum tf is 1, 10, 100, and 1000, respectively.*

Q2. (*25 marks*)

The *cluster pruning* method is introduced in Chap 7.1.6 of [MRS08].

(1) Consider the basic method (i.e., only using the closest leader to the query q). Justify the choice of choosing $\sqrt{N}$ leaders in the preprocessing step. (Hint: try to design a simple model to estimate the query processing cost)

(2) Find a minimal example where the basic method fails to return the closest document vector to the query q. You only need to give the document vectors and the query vector, and list the document returned by the cluster pruning method and the correct answer. Will the variation of the basic method (i.e., $b_1, b_2 > 1$) eliminate such problem (and guaranteed to return the correct answer)?

(3) Can you propose some modification to this method such that it guarantees returning the closest vector for any query? Describe your method and illustrate it with a small example.

Q3. (*25 marks*)

The following list of Rs and Ns represents relevant (R) and nonrelevant (N) returned documents in a ranked list of 20 documents retrieved in response to a query from a collection of 10,000 documents. The top of the ranked list is on the left of the list. This list shows 6 relevant documents. Assume that there are 8 relevant documents in total in the collection.

R R N N N N N N R N R N N N R N N N N R

(Note that spaces above are just added to make the list easier to read)

(1) What is the precision of the system on the top-20?

(2) What is the F_1 on the top-20?

(3) What is/are the uninterpolated precision(s) of the system at 25% recall?

(4) What is the interpolated precision at 33% recall?

(5) Assume that these 20 documents are the complete result set of the system. What is the MAP for the query?

Assume, now, instead, that the system returned the entire 10,000 documents in a ranked list, and these are the first 20 results returned.

(6) What is the largest possible MAP that this system could have?

(7) What is the smallest possible MAP that this system could have?

(8) In a set of experiments, only the top-20 results are evaluated by hand. The result in (5) is used to approximate the range (6) to (7). For this example, how large (in absolute terms) can the error for the MAP be by calculating (5) instead of (6) and (7) for this query?

Q4. (*25 marks*)

Consider the documents below.

docID	document text
D_1	I don't want to go A groovy king of love You can't hurry love This must be love Take me with you
D_2	All out of love Here i am I remember love Love is all Don't tell me

(1) build a unigram query likelihood language model (LM) for each document. Assume that (i) the only preprocessing done before tokenization is to transform all letters to lower cases, and (ii) we use the Jelinek-Mercer smoothing method with $\lambda = 0.5$.

(2) show which document will be ranked first for the queries:

- Q_1: `i remember you`
- Q_2: `don't want you to love me`

(3) assume that we have a prior probability distribution over the two documents as $p(D_1) = 0.7$ and $p(D_2) = 0.3$. Will this change the ranking results of the two previous queries?

Submission Instructions

You need to write your solutions to the questions in a pdf file named `ass2.pdf`. You **must**

- include your **name and student ID** in the file, and
- the file can be opened correctly on CSE machines.

You need to show the key steps to get the full mark.

Note: Collaboration is allowed. However, each person must independently write up his/her own solution.

You can then submit the file by `give cs6714 ass2 ass2.pdf`.

Late Penalty: -10% for the first two days, and **-30%** for the following days.

$$P(Q_1|d_1) = \prod_{t \in Q_1} P(t|d) = \frac{52}{836} \times \frac{11}{836} \times \frac{6}{836}$$

$$P(Q_1|d_2) = \prod_{t \in Q_1} P(t|d_2) = \frac{62}{608} \times \frac{27}{608} \times \frac{16}{608}$$

$$P(d_1|Q_1) = P(d_1 Q_1) \cdot P(Q_1)) = P(Q_1|d) \cdot P(d) \cdot P(Q_1)$$

$$\frac{1}{2} \times \frac{1}{22} + \frac{1}{2} \times \frac{1}{38} = \frac{19 + 11}{22 \times 38}$$

COMP6714 (16S2) ASSIGNMENT 2 SAMPLE SOLUTION

Q1. (*25 marks*)

(1) The BM25 formula essentially limits the impact of tfs (the value converges when $tf \to \infty$). In our case, the scoring function is

$$score(d) \leq 6f(tf_1) + 2f(tf_2) + f(tf_3)$$

where $f(x) = \frac{3x}{2+x}$. Since $\lim_{x\to\infty} f(x) = 3$, we can find the maxscores for the terms are 18, 6, and 3.

(2) We first consider D_1, with score

$$score(D_1) = 6f(1) + 2f(1) + f(1) = 9$$

Then we consider D_2

$$score(D_2) = 6f(8) + 2f(0) + f(2) = 15.90$$

At this stage, both of them become the current top-2 results, and $\tau' = 9$. Since $3 + 6 \leq \tau'$, we only need to consider A. (hence no need to score D_4)

Driven by A, the next document to score is D_5. We need to probe the lists of B and C for D_5, and compute its score as

$$score(D_5) = 6f(3) + 2f(4) + f(2) = 16.30$$

Similarly, since now $\tau' = 15.90$.

The next document to consider is D_8

$$score(D_8) = 6f(10) + 2f(0) + f(1) = 16.00$$

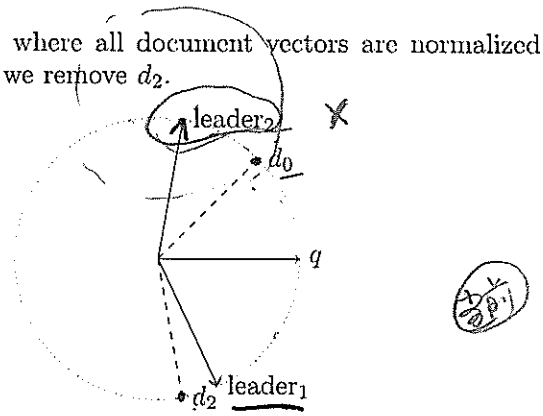
Since A's postings list is now exhausted, we conclude that the final top-2 documents are D_5 and D_8. The algorithm scored 4 documents, and accessed 10 postings.

Q2. (*25 marks*)

The *cluster pruning* method is introduced in Chap 7.1.6 of [MRS08].

(1) Let the number of leaders be x. Each leader has $\frac{b_1 N}{x} - 1$ followers on average. During the query processing, we use linear scan to find the b_2 closest leader and then find at most $b_2(\frac{b_1 N}{x})$ candidates. (We ignore the cost of finding the top-k results from these candidates) The total query processing cost (in terms of distance calculation) is $f(x) = x + b_2(\frac{b_1 N}{x})$. $\frac{\mathrm{d}}{\mathrm{d}x} f(x) = 1 - \frac{b_2 b_1 N}{x}$. Hence when $x = \sqrt{b_2 b_1 N}$, the overall query processing cost is minimized. In the basic model, $b_2 = b_1 = 1$, hence $x = \sqrt{N}$.

(2) See the following example where all document vectors are normalized to a unit vector. It is also correct if we remove d_2.



The query q is closer to $leader_1$ than $leader_2$. But the correct answer is d_0 which is a follower of $leader_2$. Even with $b_1, b_2 > 1$, we can find counter-examples (omitted).

(3) Necessary modifications:

- For each cluster c_i, calculate the maximum angle between any of the followers and the leader (denoted as θ_i).
- Assume $k = 1$. In the query processing, we first calculate the angles between all the leaders and the query. We iterate through the clusters identified by the leaders in increasing order of the angle. When visiting a cluster, we explore all its members by calculating the cosine distance to the query (essentially the angle). The stopping criteria is that the current best result has a smaller angle than α_{next}, where $\alpha_{\text{next}} = \text{angle}(c_i, q) - \theta_i$ is a lower bound of the angles between a document in c_i and the query q. This method can be easily extended to deal with top-k queries.

(The performance of the method might be heavily affected by how well documents form clusters)

Note this is just one of the correct modification methods.

Q3. (*25 marks*)

k	1	2	3	4	5	6	7	8	9	10
precision (%)	100.00	100.00	66.67	50.00	40.00	33.33	28.57	25.00	33.33	30.00
recall (%)	12.50	25.00	25.00	25.00	25.00	25.00	25.00	25.00	37.50	37.50
k	11	12	13	14	15	16	17	18	19	20
precision (%)	36.36	33.33	30.77	28.57	33.33	31.25	29.41	27.78	26.32	30.00
recall (%)	50.00	50.00	50.00	50.00	62.50	62.50	62.50	62.50	62.50	75.00

(1) precision@20 is $\frac{6}{20}$.

(2) recall@20 is $\frac{6}{8}$. $F_1 = \frac{2 \cdot \frac{3}{10} \cdot \frac{3}{4}}{(\frac{3}{10} + \frac{3}{4})} = 0.4286$

(3) 25% recall corresponds to uninterpolated precisions of 100%, 66.67%, 50.00%, 40.00%, 33.33%, 28.57%, 25.00%.
(4) the interpolated precision for 33% recall is the maximum precision achieved for $k \geq 9$. Obviously, the maximum value is $\frac{4}{11} = 0.3636$.
(5) MAP is $\frac{1}{8} \cdot (\frac{1}{1} + \frac{2}{2} + \frac{3}{9} + \frac{4}{11} + \frac{5}{15} + \frac{6}{20}) = 0.4163$.
(6) The largest possible MAP is $\frac{1}{8} \cdot (\frac{1}{1} + \frac{2}{2} + \frac{3}{9} + \frac{4}{11} + \frac{5}{15} + \frac{6}{20} + \frac{7}{21} + \frac{8}{22}) = 0.5034$.
(7) The smallest possible MAP is $\frac{1}{8} \cdot (\frac{1}{1} + \frac{2}{2} + \frac{3}{9} + \frac{4}{11} + \frac{5}{15} + \frac{6}{20} + \frac{7}{9999} + \frac{8}{10000}) = 0.4165$.
(8) $0.5034 - 0.4163 = 0.0871$

Q4. (*25 marks*)

(1) The probability distributions for each document model and the background model are:

Model		a	all	am	be	can't	don't	go
background		1/38	2/38	1/38	1/38	1/38	2/38	1/38
doc1	raw	1/22	0	0	1/22	1/22	1/22	1/22
	smoothed	30/836	22/836	11/836	30/836	30/836	41/836	30/836
doc2	raw	0	2/16	1/16	0	0	1/16	0
	smoothed	8/608	54/608	27/608	8/608	8/608	35/608	8/608

Model		groovy	here	hurry	i	is	king	love
background		1/38	1/38	1/38	3/38	1/38	1/38	6/38
doc1	raw	1/22	0	1/22	1/22	0	1/22	3/22
	smoothed	30/836	11/836	30/836	52/836	11/836	30/836	123/836
doc2	raw	0	1/16	0	2/16	1/16	0	3/16
	smoothed	8/608	27/608	8/608	62/608	27/608	8/608	105/608

Model		me	must	of	out	remember	take	tell
background		2/38	1/38	2/38	1/38	1/38	1/38	1/38
doc1	raw	1/22	1/22	1/22	0	0	1/22	0
	smoothed	41/836	30/836	41/836	11/836	11/836	30/836	11/836
doc2	raw	1/16	0	1/16	1/16	1/16	0	1/16
	smoothed	35/608	8/608	35/608	27/608	27/608	8/608	27/608

Model		this	to	want	with	you		
background		1/38	1/38	1/38	1/38	2/38		
doc1	raw	1/22	1/22	1/22	1/22	2/22		
	smoothed	30/836	30/836	30/836	30/836	60/836		
doc2	raw	0	0	0	0	0		
	smoothed	8/608	8/608	8/608	8/608	16/608		

(2)

$P(Q_1|D_1) = 52/836 * 11/836 * 60/836 = 0.0000587$

$P(Q_1|D_2) = 62/608 * 27/608 * 16/608 = 0.000119$

$P(Q_2|D_1) = 41/836 * 30/836 * 60/836 * 30/836 * 123/836 * 41/836 = 0.0000000327$

$P(Q_2|D_2) = 35/608 * 8/608 * 16/608 * 8/608 * 105/608 * 35/608 = 0.00000000261$

Thus, D_2 will be ranked first for Q_1 and D_1 will be ranked first for Q_2.

(3)

$P(Q_1|D_1) * P(D_1) = 0.0000587 * 0.7 = 0.0000411$

$P(Q_1|D_2) * P(D_2) = 0.000119 * 0.3 = 0.0000358$

$P(Q_2|D_1) * P(D_1) = 0.0000000327 * 0.7 = 0.0000000229$

$P(Q_2|D_2) * P(D_2) = 0.00000000261 * 0.3 = 0.000000000782$

Thus, D_1 will be ranked first for both queries by taking into consideration the prior.