COMP6714-Final

Chapter 1: Boolean Model

1. Example for short answer questions
   1) Why we need document frequency?
   2) List merge order (heuristic for list merge)
   3) How to put skip pointers and why?
   4) /k /S query

2. Important Content

   1) x **AND** y, x **OR** y, x **AND NOT** y, x **OR NOT** y (pseudocode and time complexity)

   2) Gallop search and its time complexity

Exercise 1.7 [⋆]

Recommend a query processing order for

d. (tangerine OR trees) AND (marmalade OR skies) AND (kaleidoscope OR eyes)

given the following postings list sizes:

Term	Postings size
eyes	213312
kaleidoscope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

SOLUTION. Using the conservative estimate of the length of unioned postings lists, the recommended order is: (kaleidoscope OR eyes) (300,321) AND (tangerine OR trees) (363,465) AND (marmalade OR skies) (379,571) However, depending on the actual distribution of postings, (tangerine OR trees) may well be longer than (marmalade OR skies) because the two components of the former are more asymmetric. For example, the union of 11 and 9990 is expected to be longer than the union of 5000 and 5000 even though the conservative estimate predicts otherwise.

S. Singh's solution

1.7Time for processing : (i) (tangerine OR trees) = O(46653+316812) = O(363465) (ii) (marmalade OR skies) = O(107913+271658) = O(379571) (iii) (kaleidoscope OR eyes) = O(46653+87009) = O(300321)

Order of processing: a. Process (i), (ii), (iii) in any order as first 3 steps (total time for these steps is O(363465+379571+300321) in any case)

b. Merge (i) AND (iii) = (iv): In case of AND operator, the complexity of merging postings list depends on the length of the shorter postings list. Therefore, the more short the smaller postings list, the lesser the time spent. The reason for choosing (i) instead of (ii) is that the output list (iv) is more probable to be shorter if (i) is chosen.

c. Merge (iv) AND (ii): This is the only merging operation left.

? **Exercise 1.4** [⋆]

For the queries below, can we still run through the intersection in time $O(x + y)$, where x and y are the lengths of the postings lists for Brutus and Caesar? If not, what can we achieve?

a. Brutus AND NOT Caesar

b. Brutus OR NOT Caesar

SOLUTION. a. Time is O(x+y). Instead of collecting documents that occur in both postings lists, collect those that occur in the first one and not in the second. b. Time is O(N) (where N is the total number of documents in the collection) assuming we need to return a complete list of all documents satisfying the query. This is because the length of the results list is only bounded by N, not by the length of the postings lists.

Exercise 1.5 [⋆]

Extend the postings merge algorithm to arbitrary Boolean query formulas. What is its time complexity? For instance, consider:

c. (Brutus OR Caesar) AND NOT (Antony OR Cleopatra)

Can we always merge in linear time? Linear in what? Can we do better than this?

SOLUTION. We can always intersect in $O(qN)$ where q is the number of query terms and N the number of documents, so the intersection time is linear in the number of documents and query terms. Since the tightest bound for the size of the results list is N, the number of documents, you cannot do better than $O(N)$.

Chapter 3: Index Construction

1. Example for short answer questions
   1) Why we need dedicated algorithms to build the index: with the limitation of hardware conditions, index construction cannot be implemented in-memory due to large corpus size.
   2) How to do in-memory index construction?

      Sorting-based algorithm or hash-based in-memory based algorithm

   3) What the problems with the immediate merge or no merge?

      Immediate merge: merge a lot, inefficient for O/S

      No merge: slow query performance.

Chapter 3: Index Construction

2. Important Content

   1)BSBI: algorithm and time complexity.
   2)SPMI: Single-pass in-memory indexing
   3)Dynamic index: immediate merge, no merge, and logarithmic merge. (how to compute complexity)

Q4. (25 *marks*)

Consider using one of the three dynamic indexing methods, namely *immediate merge*, *no merge*, and *logarithmic merge*, to build the inverted index for a collection. Let $|C|$ be the total number of bytes of the documents in the collection, M be the memory size in bytes, and B be the number of bytes in a disk block.

We only consider the total number of blocks read from or written to the disk as the I/O cost; and you can safely ignore the ceiling or floor functions in the analysis. For example, reading 1000 bytes and writing 2000 bytes from the disk costs $\frac{3000}{B}$ I/Os.

(1) How many sub-indexes will the *no merge* method create? What is the total I/O cost of indexing the collection?

(2) How many sub-indexes will the *immediate merge* method create? What is the total I/O cost of indexing the collection?

(3) How many sub-indexes will the *logarithm merge* method create (you may consider the worst case)? What is the total I/O cost of indexing the collection?

(1) No merge: It results in $\frac{|C|}{M}$ sub-indexes. Total I/O cost is:

- Reading the collection: $\frac{|C|}{B}$
- Writing: $\frac{|C|}{B}$

So total cost is $2\frac{|C|}{B}$

(2) Immediate merge: It results in 1 sub-indexes. Total I/O cost is:

- Reading the collection: $\frac{|C|}{B}$
- It performs $K = \frac{|C|}{M} - 1$ times merge; the i-th merge is between an in-memory index of size M and an on-disk subindex of size $i \cdot M$. Therefore, the totol cost is

$$\frac{1}{B} \cdot \sum_{i=1}^{K} (i * M + (i+1) * M) = \frac{M}{B} \cdot ((\frac{|C|}{M})^2 - 1)$$

So total cost is $\frac{|C|}{B} + \frac{M}{B} \cdot ((\frac{|C|}{M})^2 - 1) = O(\frac{|C|^2}{MB}) = O(\frac{|C|}{M} \cdot \frac{C}{B})$

(3) Logarithm merge: It results in $\log \frac{|C|}{M}$ sub-indexes. Total I/O cost is (assume $\frac{|C|}{M} = 2^h$ for some positive integer h)

- Reading the collection: $\frac{|C|}{B}$
- After 2^h rounds (each round builds an in-memory index of size M), there is only 1 sub-index on the disk with generation number h. Hnce it performs:
  - once merge of two generation $h-1$ sub-indexes
  - twice merge of two generation $h-2$ sub-indexes
  - ...
  - 2^{h-1} times merge of two generation 0 sub-indexes.

  Note that for simplicity and uniformity, we assume we write down conflicting index before triggering the merge (although a clever algorithm should be able to figure out the subindexes needed in the merge and perform only one merge). So the total cost is

$$\frac{1}{B} \cdot \sum_{i=0}^{h-1} 2^i \cdot (2 * 2^{h-i-1} + 2^{h-i}) \cdot M = h \cdot 2^{h+1} \cdot M = \log \frac{|C|}{M} \cdot 2 \cdot \frac{|C|}{B}$$

So the total cost is $\frac{|C|}{B} + \log \frac{|C|}{M} \cdot 2\frac{|C|}{B}$, or $O(\log \frac{|C|}{B} \cdot \frac{|C|}{B})$

(Note: it is possible to assume $\frac{|C|}{M} = 2^h - 1$ so that there is indeed h subindexes on the disk; this will only reduce the total cost by ...)

Chapter 4: Vector Space Model

1. Example for short answer questions
   1) What is / why ranked retrieval?

      Good for expert users not good for the majority of users. // Boolean model can only tell us if a document matches the query or not/ results too few or too many

      In ranked retrieval models, the system returns an ordering over the (top) documents in the collection with respect to a query.

   2) What's the problem of jaccard coefficient // raw term frequency

   3) Meaning and calculation of tf, idf?

   4) Does idf have effect on ranking one term query?

   5) Why distance is a bad idea // or why cosine similarity.

Chapter 4: Vector Space Model

2. Important Content

    1) DAAT
    2) TAAT
    3) MAXSCORE ---- MAXSCORE process

(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).

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.

Chapter 5: Evaluation

1. (very important, **一定要会**)

   https://github.com/DBWangGroupUNSW/COMP6714/blob/master/L10%20-%20EvaluationExample.ipynb
2. Question on the assignment 2.

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?

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) $0.5034 - 0.4163 = 0.0871$

Chapter 6: probabilistic model and language model

1. How to rank document? With cost and without cost.

   Slide 9-11

2. (probabilistic model) Binary independence model. (And BM25)

   What assumptions we made? 1) **Independence Assumption:** terms occur in documents independently. 2) for all terms not occurring in the query, pi = ri. 3) boolean representation of documents/queries/relevance 4) document relevance value are independent.

   How to derive? (要会推导, 很重要！！) 同时背RSV 公式。

3. Language model (n-gram)**（很重要, slides 48-51 以及54页的例题）**

Let $\vec{x}$ be the binary term incidence vector representing document D, $O(p)$ be the odd ratio of probability p, Q be the query, R and NR stand for "relevant" and "non-relevant", respectively, V is the vocabulary.
In addition, we use the shorthand notations: $p_i = p(x_i = 1|R, Q)$ and $r_i = p(x_i = 1|NR, q)$.

$$O(R|Q,\vec{x}) = \frac{p(R|Q,\vec{x})}{p(NR|Q,\vec{x})} \tag{1}$$

$$= \frac{p(R|Q)}{p(NR|Q)} \cdot \frac{p(\vec{x}|R,Q)}{p(\vec{x}|NR,Q)} \tag{2}$$

independence assumption

$$= O(p(R|Q)) \cdot \prod_{i=1}^{|V|} \frac{p(x_i|R,Q)}{p(x_i|NR,Q)} \tag{3}$$

$$= O(p(R|Q)) \cdot \prod_{x_i=1} \frac{p(x_i=1|R,Q)}{p(x_i=1|NR,Q)} \cdot \prod_{x_i=0} \frac{p(x_i=0|R,Q)}{p(x_i=0|NR,Q)} \tag{4}$$

$$= O(p(R|Q)) \cdot \prod_{x_i=1} \frac{p_i}{r_i} \cdot \prod_{x_i=0} \frac{1-p_i}{1-r_i} \tag{5}$$

when $q_i \geq 0$, $p_i > r_i$

$$= O(p(R|Q)) \cdot \prod_{x_i=1, x_i \in Q} \frac{p_i}{r_i} \cdot \prod_{x_i=0, x_i \in Q} \frac{1-p_i}{1-r_i} \tag{6}$$

$$= O(p(R|Q)) \cdot \prod_{x_i=1, x_i \in Q} \frac{p_i}{r_i} \cdot \left(\frac{\prod_{x_i \in Q} \frac{1-p_i}{1-r_i}}{\prod_{x_i=1, x_i \in Q} \frac{1-p_i}{1-r_i}} \right) \tag{7}$$

$$= O(p(R|Q)) \cdot \prod_{x_i=1, x_i \in Q} \frac{p_i(1-r_i)}{r_i(1-p_i)} \prod_{x_i \in Q} \frac{1-p_i}{1-r_i} \tag{8}$$

(a) State and *justify briefly* the assumptions made to derive Equations (3) from (2) and Equation (6) from (5) in the Binary Independence Model.

(b) State which values need to be estimated for a document collection in the final Equation (8) (i.e., other parts can be discarded safely without affecting the ranking).

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?

(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.

Chapter 7: learning to rank

1. Why weren't early attempts very successful / influential? (see slide 12)
2. Why wasn't ML much needed before and why is ML needed now?

   (see slides 13 - 14)

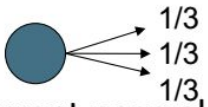
3. Learning to Rank (The ranking SVM) (see slides 23 - 24)

Chapter 8: Link Analysis

1. The concept of PageRank/ and how it is calculated in practice/ Why is it relevant for web search

PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.

- Imagine a browser doing a *random walk* on web pages:
  - Start at a random page



  - At each step, go out of the current page along one of the links on that page, equiprobably
- "In the steady state" each page has a long-term visit rate - use this as the page's score.

2. How do we compute the pagerank scores. (slides 23-24)

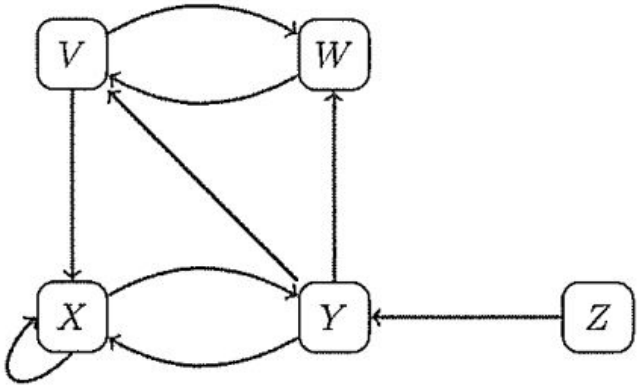
3. What problems may PageRank meet with?

1) Spider trap: definition and solution（random teleports）
2) Dead end: definition

4. How to compute steady-state?

Question 10

(10 marks)



(a) Explain the concept of PageRank, and how it is calculated in practice.

(b) Why is it relevant for Web search?

(c) Give, and briefly explain, the corresponding matrix notation of the PageRank computation.

(d) Show the final matrix that will be used for the PageRank calculation for the above graph, if the random teleporting probability is 0.2.

(e) Perform two iterations starting from the initial probability distribution vector of $(0.2, 0.2, 0.2, 0.2, 0.2)$.

Chapter 9: language models

1. What is language model? (Slide 3)
2. What is perplexity? (slide 34, the smaller the better)
3. Add-one smoothing (laplace smoothing)
4. Definition of Backoff and interpolation. (slide 59)
5. Based on interpolation, how to set the lambdas? (slide 61)
6. Understand “stupid backoff” (slide 64) and Kneser-Ney smoothing (slides 68-74)

Chapter 10: Vector Semantics

1. Four kinds of vector models. (slide 7 of part I)
2. Why we need PPMI or the problem with raw count? (slide 19 of part I)
3. What are the the definitions of PMI and PPMI? (slides 20-21)

   (公式要背下来)

4. Add-one smoothing and weighting PMI.
5. Why dense vectors? (slide 4 of part II)
6. Describe the skip-gram algorithms (slide 11 of part II)