Web Retrieval and Mining

Programming Assignment 1

資工四 黃柏智
b01902068

執行:

chmod a+x ./compile.sh
chmod a+x ./execute.sh
./compile.sh
./execute.sh [-r] -i {query-file} -o {output file} -m {model-dir} -d {NTCIR-dir}

VSM:

我把 document 視為一個 vector。我在讀取 inverted-file 時，能夠知道某個 document 包含什麼詞。在某個 document 出現的詞，會存在一個 array 中，這個 array 就是 document 的 vector，而每一個詞就是 document vector 的其中一個維度。(這個 array 裡面存的每個詞，我會給他一個 ID，這個 ID 是根據 vocab.all 裡面詞語的排列順序來設定的，我會對 array 用 ID 做 sort，之後會省下不少時間)

剛說到每個維度就是 document 中包含的詞語 (unigram 與 bigram 我都有做)，詞語存在 array 中，而 array 的每一格，還存有 tf (term-frequency) 以及 df (document-frequency)。計算維度的值是利用 tf-IDF 的公式，其中 tf 做了 Okapi / BM25 tf normalization，IDF 則做了 smoothing。詳細公式如下：

Okapi/BM25 TF = (k+1)*TF / (TF+k(1-b+b*doclen/avgdoclen))
IDF(t) = log (N + 0.5) / (DF + 0.5)

k = 2.7
b = 0.985
TF = term frequency
N = total number of docs
DF = document frequency

至於 query，我也是把它視為一個 vector。作法與 document 類似，把 query 中出現的詞彙當成 vector 的維度。我是直接用 tf (term-frequency) 當成維度的值，我試過用 tf-IDF，效果不好，或許跟 query 的平均大小跟 document 的平均大小差太多有關。

在 query 中，包含 title、question、narrative、concept 四部分。但在幾次實驗中發現 narrative 的結果太糟，我直接去看 narrative 的內容後發現，有太多冗餘詞彙，因此後來直接忽略 narrative。

現在 query 剩下 title、question、concept 三部分，我對三部分施予不同權重，原本我給 title 和 concept 最高，卻發現 title 權重越低，分數會越高，最後 title 權重直接設為 0。

最後的權重如下：
<title> 權重 = 0
<question> 權重 = 1
<concept> 權重 = 1.52

此外，我發現 question 中有些贅字切掉後，分數更高。贅字如下：
- 查詢
- 有關
- 以及
- 與
- ，
- 、
- 。

而 concept 中的標點符號我也全部切掉：
- 、
- 。

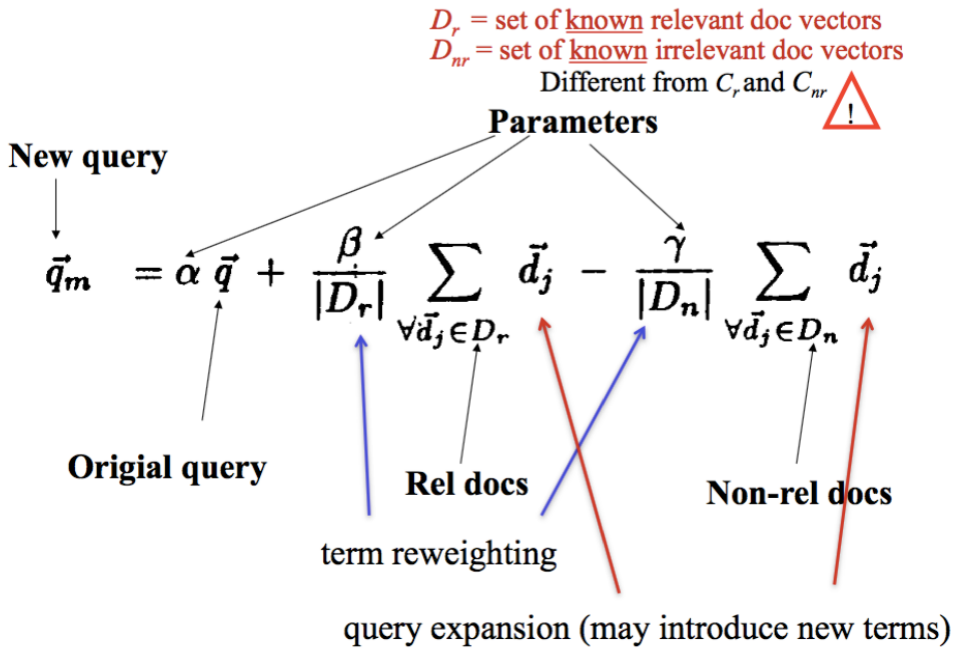
當 document 與 query 的 vector 都處理完畢，就能開始算 similarity，我一開始用 Dot product similarity，就是直接將雙方的 vector 做內積，但無法突破 baseline。後來採用 Cosine similarity，分數就直接衝到 0.7X 了。公式如下：

$$sim(\vec{Q},\vec{D}_i) = \frac{\sum_{j=1}^{N} w_{qj} * w_{ij}}{\sqrt{\sum_{j=1}^{N}(w_{qj})^2 * \sum_{j=1}^{N}(w_{ij})^2}}$$

最後用算出來的 similarity 挑出 top100 documents。

Rocchio Relevance Feedback:

根據上面的方法算出 top documents 之後，挑出前幾名 document，將他們視為跟 query 非常相關，用這些 document 去修改 query vector 的值。
我會找尋前幾名 document，當作 relevance docs。接著逐一查看在 relevance docs 中的所有詞彙，看看這些詞彙是否也有在 query 中出現。若有，將該詞彙在 query vector 中代表的維度的值提升。至於要提升多少，我是直接將該詞彙在 document 中的 tf-IDF 加入 query 的該維度值中。



經過實驗，得出的 parameters 如下：
relevant doc: 取前 9 名的 document
alpha = 0.9
beta = 0.021

Results of Experiments:

• Map value under different parameters of VSM
首先只測 VSM 做出的結果。因為我認為現在的值在加入 Rocchio relevance feedback 後會改，現階段做出的最佳解不一定是之後的最佳解，只能當作下階段的調整依據的大方向，因此沒有花很多時間在這裡。
以下是對 Okapi / BM25 TF Normalization 中參數 k, b 的測試：

k	b	MAP score
2.23	1.02	0.795256298517
2.23	1.01	0.795294513996
2.23	1.00	0.7953
2.23	0.99	0.7960
2.23	0.98	0.795663904799
2.23	0.97	0.795572972343
2.23	0.96	0.795686692052
2.25	1.02	0.795209833666
2.25	1.01	0.795303833536

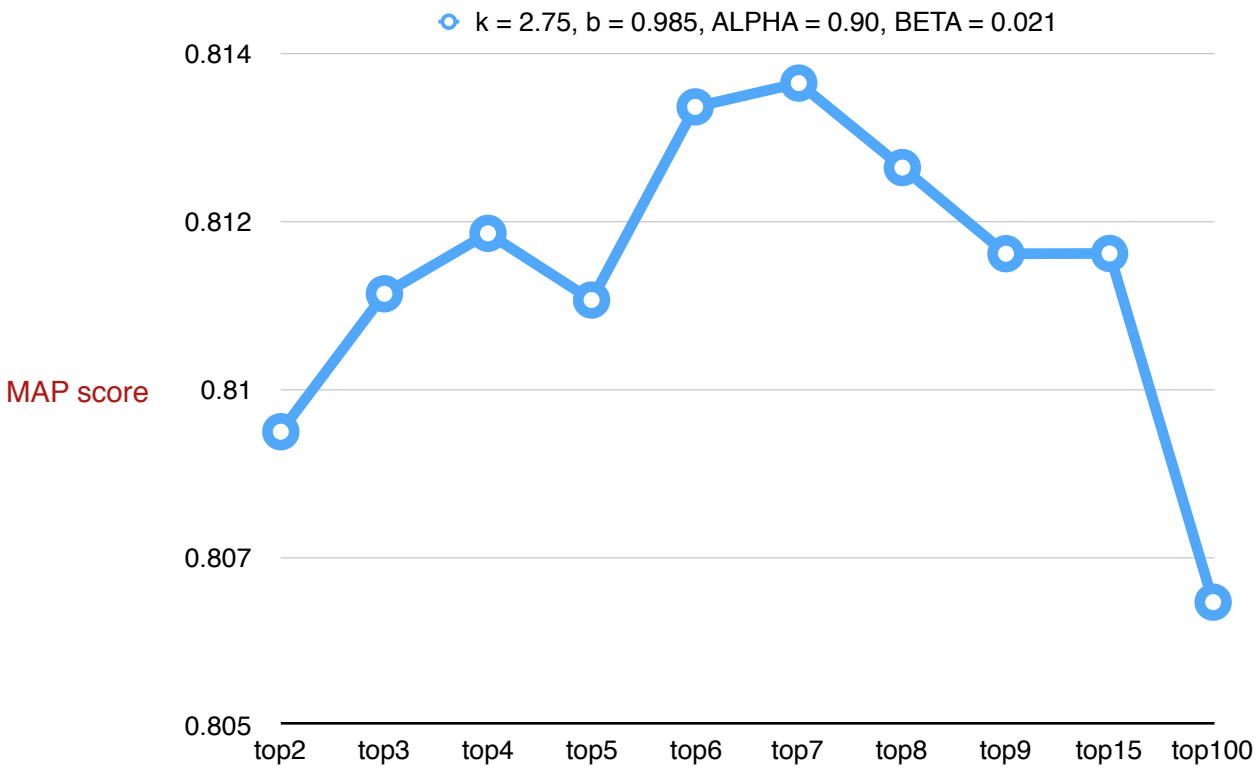
• Feedback vs No-Feedback

加上 relevance feedback 後，有 k, b, alpha, beta 四個變數可以測試，其中 alpha, beta 是 relevance feedback 中的變數，以下為測試結果：

k	b	alpha	beta	MAP score
2.75	1.00	0.86	0.0253	0.800086942691
2.75	0.99	0.86	0.0253	0.808238386901
2.75	0.99	0.88	0.0253	0.808470880362
2.75	0.99	0.89	0.0253	0.808471998523
2.75	0.99	0.895	0.0253	0.808471998523
2.75	0.99	0.90	0.0253	0.808471998523
2.75	0.985	0.90	0.0253	0.808482874806
2.8	0.985	0.90	0.0253	0.807052254986
2.7	0.985	0.90	0.0253	0.808021020087
2.75	0.985	0.90	0.022	0.808212078565
2.75	0.985	0.90	0.0215	0.808261266354
2.75	0.985	0.90	0.02125	0.808874665902
2.75	0.985	0.90	0.021	0.808929322404
2.75	0.985	0.90	0.0205	0.808731189251
2.75	0.985	0.90	0.02	0.808801022588
2.75	0.985	0.90	0.018	0.808738330526

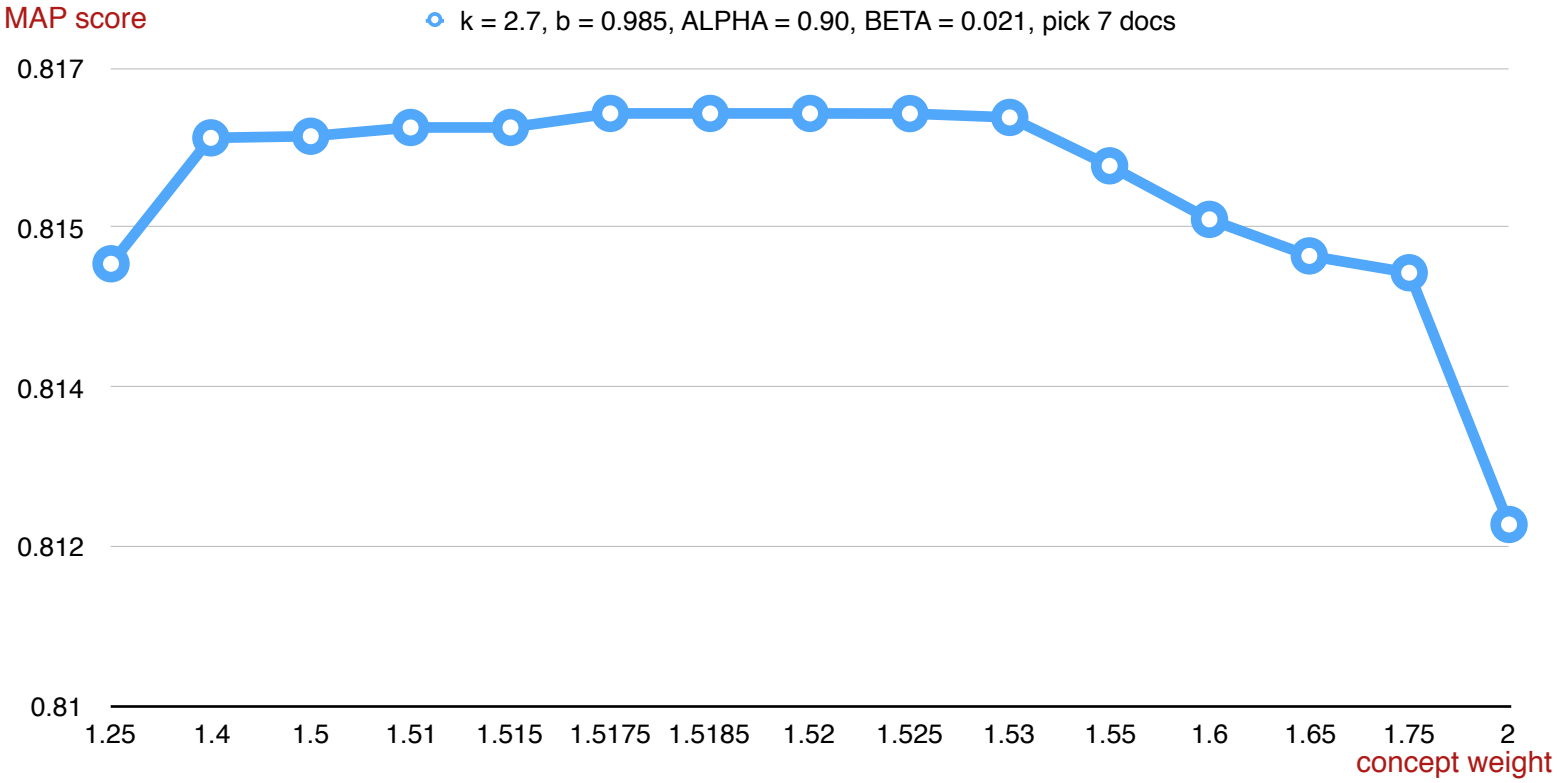
接著我選擇 k = 2.75, b = 0.985, ALPHA = 0.90, BETA = 0.021 去找出 relevance feedback 需要挑前幾個 document 比較好，以下是測試結果：

可得知取前 7 個 document 來做 feedback 是最佳的。

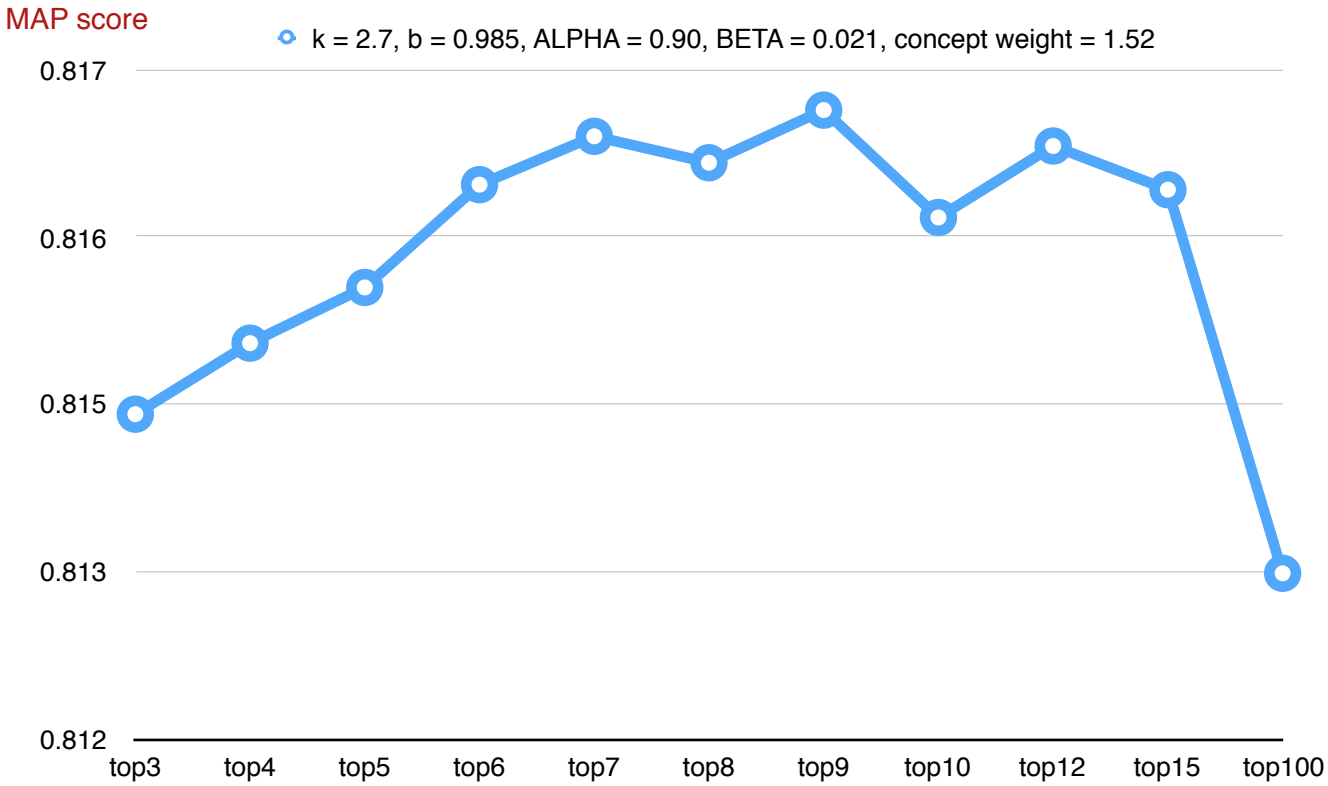


• Other experiments

在做完 relevance feedback 後，我開始針對 query 中 <question>、<concept>，做不同權重測試，我把 question 都設為 1，只調整 concept 權重。以下是測試結果：
最後得出最佳 concept 的權重為 1.52



在得到此權重後，我又重新算了一次 feedback 要挑前幾篇 documents 會拿到最棒的分數，以下是測試結果：
現在反而要取前 9 個 document 來做 feedback 才會最好



接著，我就重複做著找 question & concept 的最佳權重與找最佳 feedback document 數目，直到某次找最佳 feedback document 數目時，結果保持在取 10 個 document，不再變動了。此時的參數如下：
k = 2.7, b = 0.985, alpha = 0.9, beta = 0.021, concept weight = 1.49

此外，我還測試了如果跑完 feedback 後再跑一次 feedback 結果會如何，
但不管參數怎麼設，MAP score 都往下降大約 0.01，因此最後得出只做一次 feedback 是最佳解。

我也有測試只用 bigram，以及同時使用 unigram+bigram。
後者的 performance 大概都能贏前者 0.1 以上。這樣在 scoreboard 上就差十幾名了。

Discussions:

- 這次作業中，參數的設置佔了非常重要的角色。能嘗試越多種參數組合，就越有機會拿到高分。神奇的是，有時你調了 A 參數，分數下降，你改了 B 參數。分數下降，但你同時改變兩個參數，分數卻上升了，實在令人哭笑不得。

- 乘上，由於需要測許多參數組合，程式跑得越快，就能試更多次。我用 C 寫，跑一次大約需要 50 秒，聽到有些同學用 python，還沒做 relevance feedback 就要 5 分鐘，做 relevance feedback 就快要 10 分鐘，等於說他試一次參數，我能試 10 次，差距就出來了。

- 在 relevance feedback 階段，根據公式，我應該要扣掉不相關的 document，但你算完 similarity 後，你對相關的 document 的信心指數會大過不相關的 document。會有這種現象，或許是因為在這次作業中，會造成不相關的原因，就是 document 用的詞彙跟 query 不同，但不同的兩個詞卻可能是同義詞，在這次作業的方法中純粹針對 similarity，無法判斷同義詞 (VSM 認定所有維度都是獨立的)，也無法偵測語意相似度。因此在你不確定是否不相關的情況下，選擇忽略扣除的動作，會讓你的 performance 更佳。
  至於那些做出來相關的 document，因為用了許多跟 query 一樣的詞彙，因此有可能真的是相關，但也無法完全肯定，因此給予的權重不能太高。

- 在處理 query 時，把贅字去掉，score 可能會上升。我一開始先把 '相關'、'查詢'、'有關' 這些明顯的贅字去除，score 的確上升了。之後，我認為 query 中的 '與'、'之'、'所' 也應該去除，但這三的字只有 '與' 去掉分數會上升。這個情況可能是因為，在 '與' 前後的兩個詞彙關聯性比較小，指的可能是兩件天差地遠的事件，而 '之'、'所' 前後的詞彙通常會有某種意義上的關係。同樣的情況也發生在 '的' 身上。