Web Retrieval and Mining Spring 2017 Programming HW1

B02902030 資工四 曹燁文

一、VSM

建立向量時，只使用query有的term(unigram+bigram)，可以有效減少計算量，因為若不使用只有含query term的那幾維向量，便會出現許多零的entry，對於計算similarity毫無貢獻。

另外query只用了concept這個tag，實驗發現加入如title會讓結果變差。narrative雖然有更完整的資訊，例如某些關鍵字提供資訊說哪些文件是不相關的，但我想這已經牽涉到語意分析，超出這次的範圍，但若能好好利用定能讓結果非常好。

使用Okapi/BM25 TF Normalization，參數使用見第三部分的實驗。query的doclen/avgdoclen我直接設為1，因為我認為query是一個獨立於documents的文件，且同時只有一份，所以可以直接設為1。document的doclen/avgdoclen是直接找文件中被tag ... 夾住的部分，並計算長度與平均長度。

二、Rocchio Relevance Feedback

參數見第三部分的實驗，定義relevant的方式為先用VSM算出第一次的排序結果，取前10個作為relevant，non-relevant的話是取排序結果第10000名之後的1000個。我有嘗試連續兩次做Rocchio，但結果沒有變好，所以最後只使用一次Rocchio。

FB社團有提到使用Rocchio可能會讓結果變差，我在train上進步、test上退步，我想有些原因可能是第一次的排序結果並不準確，另外一個原因我猜測是relevant/non-relevant的群集不夠集中，或是不具有代表性，以至於不能讓query調整到更好的方向，例如說好的non-relevant應該要跟relevant反方向，但像我的作法只取query有的term建立向量，便無法產生出與relevant反方向的向量，我想可以使用全部的term建立向量，這樣也許可以讓結果更好，但計算量會十分龐大。

三、Experiment Result

首先比較不同Okapi參數的差異，再比較Rocchio不同參數的差異

Okapi:

okapi_k, okapi_b	2.0, 0.75	1.2, 0.75	3.0, 1.0
MAP (train)	0.765	0.756	0.775
MAP (test)	0.762	0.752	0.77311

Rocchio: (Okapi參數固定為(3.0, 1.0), non-relevant為第二部分說的, rel_len是取前幾個第一次排序完的結果)

α, β, γ, rel_len	0.8, 0.2, 0, 10	0.79, 0.2, 0.01, 10	0.79, 0.2, 0.01, 15	0.79, 0.2, 0.01, 5	0.78, 0.2, 0.02, 10
MAP (train)	0.7849	0.7849	0.7777	0.7774	0.7850

α, β, γ, rel_len=10	0.78, 0.2, 0	0.9, 0.1, 0	1, 0.1, 0.1	1.13, 0, 0.13	1.2, 0, 0.2
MAP (test)	0.7642	0.7705	0.7714	0.7722	0.77110

四、Discussion

在這次作業中，第一次實作了VSM以及Rocchio，對於這簡單卻有效的構想非常佩服，可惜的是VSM無法做出語意的分析，否則可以判斷出許多文章雖然有重要的關鍵字，卻不是query真正希望的答案。

另外，實作之後也發現加入Rocchio真的不一定會變好，也許是參數沒有調好，也有可能是像我第二部分說明的那樣，是建立向量的方式不好。但是我覺得Rocchio是個很直觀、漂亮的方法，就算結果不好但演算法精神值得學習。

除此之外我想過許多其他作法，但時間不足無法一一實作，以下大致說明我的一些想法：

1. 用document之間的相關性讓結果更好：這部分的想法與Rocchio有些相似，不同的地方在於沒有要修改query，而是希望比對答案與非答案文件的相關性，看看是否能找出蛛絲馬跡。然而實驗發現train給的答案與自己產生的答案最後算出來的similarity非常接近，所以此方法失敗。
2. 調參數：有非常多可以調參數的地方，例如說每個query的Okapi值可以不同、Rocchio的參數也可以根據query不同而有所改變，另外上面雖提過做了兩次Rocchio變差，但也許是參數問題，可以讓relevance/non-relevance的數量做變化、前面三個參數不要固定，又或是不要用平均，而是按照排名給予加權。
3. 機器學習: 有粗略想過透過機器學習的方式，學出Aq+b = q'裡的A,b，q是原先的query，q'是新的query，A, b分別是轉換矩陣和bias，也就是透過學習讓機器自動學出如何加工query讓結果更好。但這有兩個問題：若是像我用query內的term當作向量的話，每次的長度都不同，除非每次query都做降維等方式讓向量長度相同，否則無法學習。第二個問題是資料太少，這次作業只有10+20筆query，根本無法進行足夠的學習，所以這個方法貌似不可行。