COMP6714 ASSIGNMENT 2

Changjian Zhu   z5082423

Q1.

1) $maxscore(d_{max},\ \{t\}) = \sum_{t \in Q} idf_t \cdot \frac{3tf_{t,d}}{2+tf_{t,d}} \frac{3tf_{t,Q}}{2+tf_{t,Q}} = idf_t \cdot \frac{3max_{tf}}{2+max_{tf}}$

a. If the maximum tf in docs is 1, $maxscore(A) = 6 \cdot \frac{3\cdot1}{2+1}\frac{3\cdot1}{2+1} = 6$,
$score(d,\ B) = 2,\ score(d,\ C) = 1$

b. If the maximum tf in docs is 10, $maxscore(A) = 6 \cdot \frac{3\cdot10}{2+10}\frac{3\cdot1}{2+1} = 15$,
$score(d,\ B) = 5,\ score(d,\ C) = 2.5$

c. If the maximum tf in docs is 100, $maxscore(A) \approx 6 \cdot \frac{3\cdot100}{2+100}\frac{3\cdot1}{2+1} = 18$,
$score(d,\ B) \approx 6,\ score(d,\ C) \approx 3$

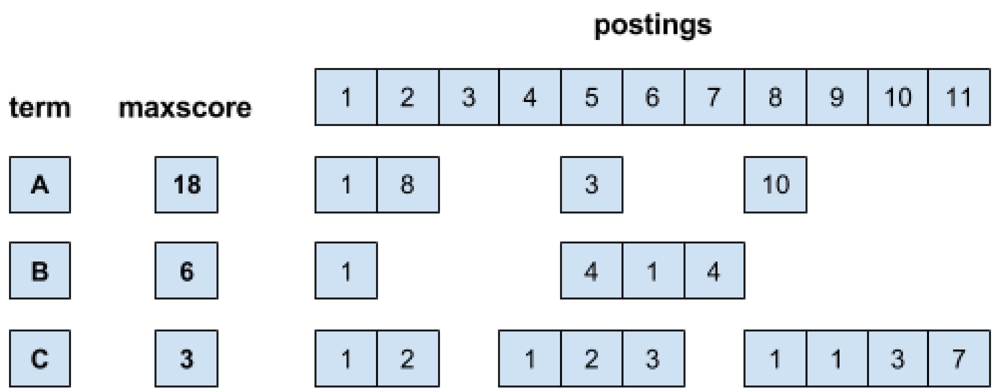
d. If the maximum tf in docs is 1000, $maxscore(A) \approx 6 \cdot \frac{3\cdot1000}{2+1000}\frac{3\cdot1}{2+1} = 18$,
$score(d,\ B) \approx 6,\ score(d,\ C) \approx 3$

Got it :), So when tf becomes infinitely great:

$maxscore(d_{max},\ \{t\}) = idf_t \cdot \frac{3max_{tf}}{2+max_{tf}} \approx 3 \cdot idf_t$

Hence, the maxscore for each keyword is 18(A), 6(B) and 3(C).

2)



Assume the top-2 result is tuple TOP2.

Step 1: When the cursor is on doc2, TOP2 = (15.9, 9)

Step 2: skip D_4 simply because its $maxscore_c < R_{min}$, eg. 3 < 9.

Step 3: D_5 contains all 3 terms, $maxscore_a + maxscore_b + maxscore_c > R_{min}$

So the score of D_5 is 16.6, then we have to update TOP2 to (16.6, 15.9)

Step 4: skip doc6 and doc7

Step 5: The score of D_8 is 16, push 16 and pop, then TOP2 = (16, 16.6)

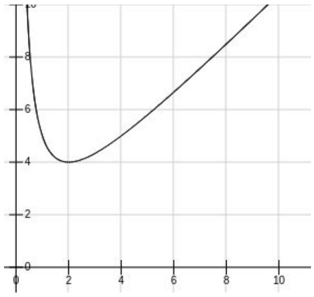
Step 6: Skip D_9, D_{10} and D_{11}

Step 7: According to TOP=(16, 16.6), so the top-2 results is D_5 and doc8

Q2

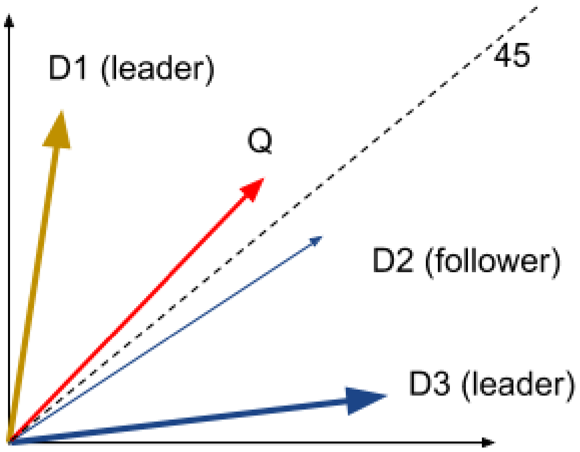
1) For each query (assume there are x leaders, N documents):
   Step1: find leader based on cosine similarity, so the cost is $O(x)$.
   Step2: computing cosine similarity for each follower, the cost is $O(N/x)$.

   The total query processing cost is $O(x+N/x)$.
   Visual function of the cost (N=4):



   $f(x) = x + N/x \rightarrow f'(x) = 1 - N/x^2 = ((x - \sqrt{N})(x + \sqrt{N}))/x^2$
   So it clear that $\sqrt{N}$ is the critical point and the query cost is minimum when the number of leaders is $\sqrt{N}$

2)

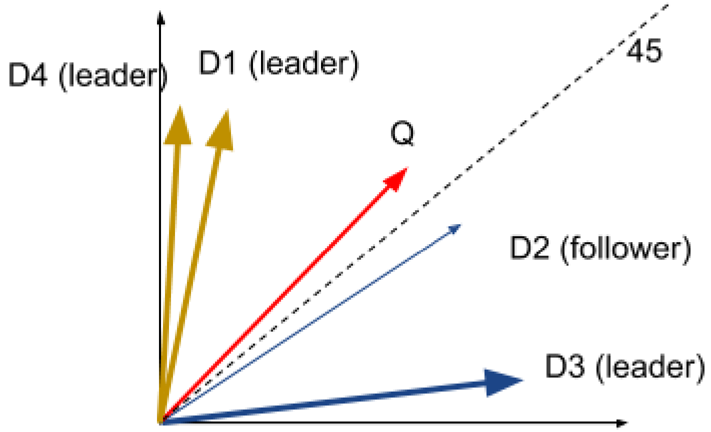


   Here is one 2-D example failing to return the closest document:
   The red line is query Q. The closest document should be D2, however, D2 belong to the cluster of D3, so query can never reach it.

   In my example, the problem does can be solved by:
   1. Changing b_2 to 2, then in the process of query, the query can reach the cluster of leader D3.
   2. Changing b_1 to 2, then in the preprocessing, D2 will both belong to D1 and D3, the query Q can find it's closest document D2.

   However, it can not 100% find the closest document if not setting b_1 or b_2 to $\sqrt{N} - 1$.

In this my example below, even b_2 is changed to 2, query can only reach the cluster of leaders D4 and D1, still fail to find closest document D2.



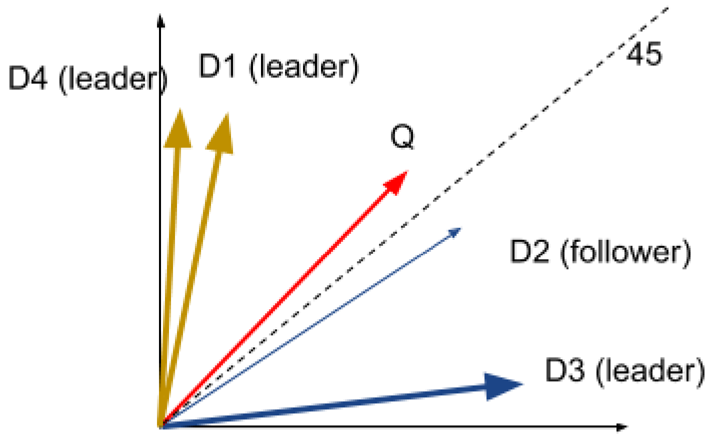
Ps. some followers are not shown to simplify the example

3)

For the reason that we only has to consider the distance(cosine) between vectors(documents), so it's actually a **1-D problem.**
When query hits **red area(cluster)**, we can absolutely get the closest vector by also checking the **left and right closest clusters.**

Here is my solution:
For each query, after finding the **leader L that is closest to query**, we also have to include left and right clusters of L:
1. **Closest Leader** and has the value **larger than L**
2. **Closest Leader** and has the value **lower than L**

For example, after getting the leader D1, it will also check the cluster of left leader D4 and right leader D3.

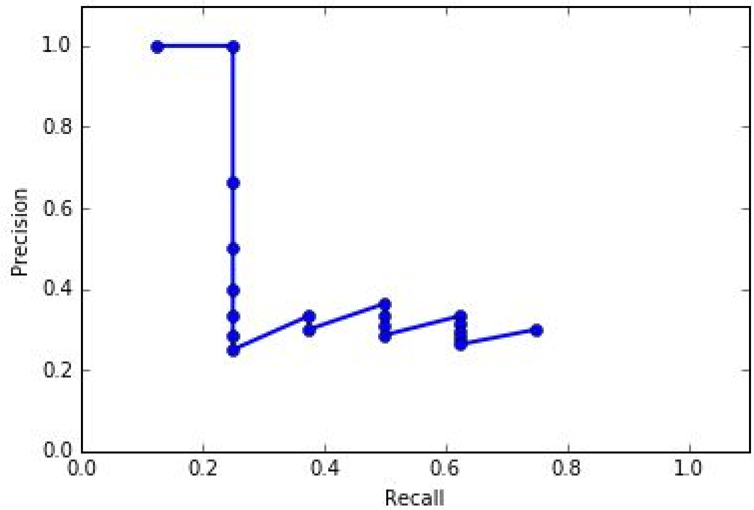


Ps. some followers are not shown to simplify the example

Q3

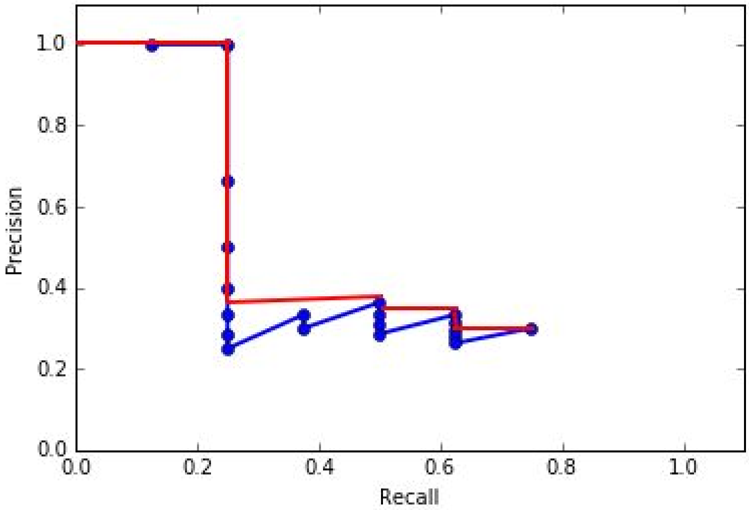
1) $Precision = P(relevant|retrieved)$ = $(relevant\ docs\ in\ top20)\ /\ 20 = 6/20 = 30\%$
2) $Recall = P(retrieved|relevant)$ = $(relevant\ docs\ in\ top20)\ /\ (relevant\ docs\ in\ total) = 6/8 = 75\%$
   $F_1 = \frac{1}{1/(2R)+1/(2R)} = \frac{2RP}{R+P} = 0.429$
3) See the table below

k-th Output	Judgement	Prec	Recall	k-th Output	Judgement	Prec	Recall
1	R	1/1	1/8	11	R	4/11	4/8
2	**R**	**2/2**	**2/8**	12	N	4/12	4/8
3	**N**	**2/3**	**2/8**	13	N	4/13	4/8
4	**N**	**2/4**	**2/8**	14	N	4/14	4/8
5	**N**	**2/5**	**2/8**	15	R	5/15	5/8
6	**N**	**2/6**	**2/8**	16	N	5/16	5/8
7	**N**	**2/7**	**2/8**	17	N	5/17	5/8
8	**N**	**2/8**	**2/8**	18	N	5/18	5/8
9	R	3/9	3/8	19	N	5/19	5/8
10	N	3/10	3/8	20	R	6/20	6/8



So the uninterpolated precision(s) of the system at 25% recall are [2/2, 2/3, 2/4, 2/5, 2/6, 2/7, 2/8]

4) Interpolated precision(red lines):



So the interpolated precision at 33% recall is 4/11(k=11).

5) 1. MAP: Average of the precision value
2. Average precisions essentially only count precisions at positions where a relevant document is retrieved.
3. Relevant documents in total: 8.
$MAP = (1/1 + 2/2 + 3/9 + 4/11 + 5/15 + 6/20)/8 = 0.416$

6) We will get the largest MAP if the rest two relevant documents appear at the position that k equals to 21 and 22.
$Largest\ MAP = (1/1 + 2/2 + 3/9 + 4/11 + 5/15 + 6/20 + 7/21 + 8/22)/8 = 0.503$

7) We will get the smallest MAP if the rest (9800) documents contain relevant documents in the last two documents, e.g. 9799 and 9800.
$Smallest\ MAP = (1/1 + 2/2 + 3/9 + 4/11 + 5/15 + 6/20 + 1/9799 + 1/9799)/8 = 0.416$

8) Why The result in (5) is used to approximate the range (6) to (7).
Shouldn't it be the lowest MAP that can never be achieved. Sorry I give up.
Guessing the answer of this question is $Largest\ MAP - Smallest\ MAP = 0.0871$

Q4

1) In my understanding, unigram query likelihood language **model** is the just the possibility of each term:

The individual-document model for D_1 ,
[('love', '3/22'), ('you', '2/22'), ('hurry', '1/22'), ('be', '1/22'), ('must', '1/22'), ('with', '1/22'), ('groovy', '1/22'), ('i', '1/22'), ('take', '1/22'), ('want', '1/22'), ('to', '1/22'), ('can't', '1/22'), ('me', '1/22'), ('this', '1/22'), ('a', '1/22'), ('go', '1/22'), ('don't', '1/22'), ('king', '1/22'), ('of', '1/22')]

The individual-document model for D_2
[('love', '3/16'), ('i', '2/16'), ('all', '2/16'), ('don't', '1/16'), ('out', '1/16'), ('remember', '1/16'), ('me', '1/16'), ('here', '1/16'), ('am', '1/16'), ('is', '1/16'), ('of', '1/16'), ('tell', '1/16')]

Collection/background language model or D_1 and D_2

[('love', '6/38'), ('i', '3/38'), ('all', '2/38'), ('me', '2/38'), ('don't', '2/38'), ('of', '2/38'), ('you', '2/38'), ('a', '1/38'), ('groovy', '1/38'), ('remember', '1/38'), ('want', '1/38'), ('to', '1/38'), ('can't', '1/38'), ('out', '1/38'), ('go', '1/38'), ('is', '1/38'), ('tell', '1/38'), ('hurry', '1/38'), ('be', '1/38'), ('must', '1/38'), ('with', '1/38'), ('take', '1/38'), ('this', '1/38'), ('here', '1/38'), ('king', '1/38'), ('am', '1/38')]

2) Ranking formula:

$$p(d)\prod_{t\in Q}((1-\lambda)\frac{raw\ term\ frequency\ in\ doc}{total\ number\ of\ tokens\ in\ doc}+\lambda\frac{raw\ term\ frequency\ in\ doc}{total\ number\ of\ tokens\ in\ doc1\ and\ doc2})$$

For Q1 to D_1, according to the formula:

$$p(Q_1|d_1)=p("i"|d_1)p("remember"|d_1)p("you"|d_1)$$
$$=[(1/22+3/38)/2]x[(0/22+1/38)/2]x[(2/22+2/38)/2]$$
$$=5.873926e-05$$

I wrote a piece program to calculate it:

```
doc1 = '''I don't want to go A groovy king of love
          You can't hurry love This must be love Take me with you '''.lower()
doc2 = '''All out of love Here i am
          I remember love Love is all Don't tell me '''.lower()

tokens_len_d1, tokens_len_d2 = len(doc1.split()), len(doc2.split())
tokens_len_all = tokens_len_d1 + tokens_len_d2
lm_d1, lm_d2 = Counter(doc1.split()), Counter(doc2.split())
lm_all = lm_d1 + lm_d2

def count_ranking(LM, tokens_len, query, delta=0.5):
    ranking = 0
    for term in query.split():
        p = (1 - delta) * (LM[term] / tokens_len) \
            + delta * (lm_all[term] / tokens_len_all)
        ranking = p if ranking == 0 else ranking * p
    return ranking

query_1 = "i remember you"
print("P(Q1|d1): %e  (manual)"%(((1/22+3/38)/2) * ((0/22+1/38)/2) * ((2/22+2/38)/2)))
print("P(Q1|d1): %e"%count_ranking(lm_d1, tokens_len_d1, query_1))
print("P(Q1|d2): %e\n"%count_ranking(lm_d2, tokens_len_d2, query_1))

query_2 = "don't want you to love me"
print("P(Q2|d1): %e"%count_ranking(lm_d1, tokens_len_d1, query_2))
print("P(Q2|d2): %e"%count_ranking(lm_d2, tokens_len_d2, query_2))
```

```
P(Q1|d1): 5.873926e-05  (manual)
P(Q1|d1): 5.873926e-05
P(Q1|d2): 1.191694e-04

P(Q2|d1): 3.270611e-08
P(Q2|d2): 2.607377e-09
```

So D_2 will be ranked first for Q_1, and D_1 has a better ranking for Q_2.

3) Here are the new ranking results:

```
query_1 = "i remember you"
print("P(Q1|d1): %e"%(count_ranking(lm_d1, tokens_len_d1, query_1)*0.7))
print("P(Q1|d2): %e\n"%(count_ranking(lm_d2, tokens_len_d2, query_1)*0.3))

query_2 = "don't want you to love me"
print("P(Q2|d1): %e"%(count_ranking(lm_d1, tokens_len_d1, query_2)*0.7))
print("P(Q2|d2): %e"%(count_ranking(lm_d2, tokens_len_d2, query_2)*0.3))
```

```
P(Q1|d1): 4.111748e-05
P(Q1|d2): 3.575082e-05

P(Q2|d1): 2.289428e-08
P(Q2|d2): 7.822132e-10
```

So for both queries, $Ranking : D_1 > D_2$

Thanks for your patience :)

end