COMP6714 Assignment 1

Chencheng Xie
sid: z5237028

November 2019

Q1. (1) The algorithm for Intersection(A, B)

Algorithm 1: Intersect(A, B)

```
   Input: unsorted lists A, B
   Output: intersection of two list I
1  if A.Len == 0 or B.Len == 0 then
       /* If any list is empty, no intersection can be found, return []   */
2      return [ ];
3  else
       /* If both list contains only one element                          */
4      if A.Len == 1 and B.Len == 1 then
5          if A[0] == B[0] then
               /* return list contain this element if they match          */
6              return [ A[0] ];
7          else
               /* return empty list, otherwise                            */
8              return [ ];
9      else
           /* Recursive Branch                                            */
10         threshold = A[0];
11         A1, A2, B1, B2 = [ ];
           /* divide A,B into A1, A2, B1, B2 by ''smaller or equal to'' or
              ''larger than'' threshold value                             */
12         foreach a ∈ A do
13             if a ≤ threshold then
14                 A1 += [a];
15             else
16                 A2 += [a];
17         foreach b ∈ B do
18             if b ≤ threshold then
19                 B1 += [b];
20             else
21                 B2 += [b];
           /* Return the concatenation of two intersections               */
22         I = Intersect(A1, B1) + Intersect(A2, B2);
23         return I;
```

(2) Think of a method to divide each input list into k sub-lists.
We can divide each input into k sub-lists ($k \geq 2$): $(A_1, A_2, ..., A_k)$ & $(B_1, B_2, ..., B_k)$, each sub-lists will be smaller compare to how we break lists into two parts in (1). Smaller size of lists will reach BASE CASE of recursive function faster, and we only need to return the concatenation of "**Intersect**(A_1, B_1), **Intersect**(A_2, B_2), ..., **Intersect**(A_{k-1}, B_{k-1}), **Intersect**(A_k, B_k)".

So the only difference is that we need to select $k-1$ threshold values to have k intervals to separate list into $k - list$. We can easily choose the first $k-1$ elements as threshold value. The performance varies with respect to how well the list is divided. If the list is evenly divided, the performance can be $O(m \log_k m + n \log_k n)$. If the list is badly divided, each time selected threshold values are top $k-1$ values or bottom $k-1$ values, the performance can be $O(m^2 + n^2)$. So the performance ranges between $O(m \log_k m + n \log_k n) \sim O(m^2 + n^2)$.

Q2. (1) Show that if the logarithmic merge strategy is used, it will result in at most $\lceil \log_2 t \rceil$ sub-indexes.
The logarithmic-merge strategy merges **two** sub-indexes with same generation, it allows only **one** sub-index with same generation at anytime. So in the worse situation, the whole sub-indexes collection is: $I_0, I_1, ..., I_n$, and they sum up to totally t sub-indexes ($t \times I_0$) if no-merge strategy is used.
This is essentially:

$$\sum_{i=0}^{k} I_i = t \times I_0 \tag{1}$$

And also each $I_k = 2 \times I_{k-1}$, so:

$$\sum_{i=0}^{k} I_i = 2^{k+1} I_0 - I_0 = 2 \times 2^k I_0 - I_0 \tag{2}$$

Combine equation (1) & (2), we have:

$$\begin{aligned} 2 \times 2^k I_0 - I_0 &= t \times I_0 \\ 2 \times 2^k + 1 &= t \\ 2^{k+1} &= t - 1 \\ k &= \log_2(t-1) - 1 \\ k &= \log_2 t + \log_2(\frac{t-1}{t}) - 1 \\ k &= \log_2 t + 0 - 1 \approx \lceil log_2 t \rceil \end{aligned} \tag{3}$$

So the number of sub-indexes is at most $\lceil log_2 t \rceil$.
This can also be interpreted as: The size of sub-indexes increase exponentially with base 2, their sizes sum up to $t \cdot I_0$ (each I_0 has M pages), so the number of sub-indexes $k = \lceil log_2 t \rceil$.

(2) Prove that the total I/O cost of the logarithmic merge is $O(t \cdot M \cdot \log_2 t)$.
The total I/O cost of the logarithmic merge is formed by **two** parts:
1. read the whole collection into memory: $O(t \cdot M)$.
2. merge operations in the disk:
Let's consider time frame from 0 to T, where disk is **empty** at time 0 and disk has

$I_0, I_1, ..., I_k$ at time T. The time-frame T needs **two** time-frame $T-1$ plus I_0:

$$\sum_{i=0}^{k} I_i = 2 \times \sum_{i=0}^{k-1} I_i + I_0 = 2^k \times I_0 + I_0 \tag{4}$$

From **empty** to time T, each time frame roughly doubled, so we have $\log_2 t$ time frames, $(T = \log_2 t)$.
Now consider the cost of last time frame T:

each sub-indexes I_i we need **two** input of I_{i-1} and **one** output of I_i, so cost of $I_k = (2 \times 2^{k-1} + 2^k)I_0$, and I_0 has M pages, so cost of $I_k = 2 \times 2^k M = 2 \times t \cdot M$.
Sum of this time frame T is $2 \times I_k - I_0 = 2 \times 2^{k+1}M - M = 4 \times t \cdot M - M$.
Now we calculate the total cost of merge: number of time frames $\times$ the cost of worst time frame $= \log_2 t \times 4 \times t \cdot M$
And the total cost of whole I/O process is: $t \cdot M + 4 \log_2 t \cdot tM$ or $O(t \cdot M \cdot \log_2 t)$.

Q3. We can summary the following table with provided information:

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

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

$$\begin{aligned} \textit{Precision } P &= \frac{\#\ \textit{Retrieved } R}{(\#\ \textit{Retrieved } R + \#\ \textit{Retrieved } N)} \\ &= \frac{6}{20} = 30\% \end{aligned} \tag{5}$$

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

$$\begin{aligned} \textit{Recall } R &= \frac{\textit{\# Retrieved } R}{(\textit{\# Retrieved } R + \textit{\# Not Retrieved } R)} \\ &= \frac{6}{8} = 75\% \end{aligned} \tag{6}$$

$$\begin{aligned} F_1 &= \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} \\ &\quad (\textit{with } \beta = 1 \textit{ or } \alpha = 1/2) \\ &= \frac{(1^2+1) \times 30\% \times 75\%}{1^2 \times 30\% + 75\%} \\ &= 42.86\% \end{aligned} \tag{7}$$

(3) What is/are the uninterpolated precision(s) of the system at 25% recall?
All the following returns give 25% recall:

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

And the corresponding uninterpolated precisions are:

Judgements	Uninterpolated precisions	Recall
R R	2/2 (100%)	2/8
R R N	2/3 (66.7%)	2/8
R R N N	2/4 (50%)	2/8
R R N N N	2/5 (40%)	2/8
R R N N N N	2/6 (33.3%)	2/8
R R N N N N N	2/7 (28.6%)	2/8
R R N N N N N N	2/8 (25%)	2/8

(4) What is the interpolated precision at 33% recall?
The interpolated precision is the maximum precision to the right of the value.
$8th$ Output has recall 25%, $9th$ Output has recall 37.5%, so the interpolated precision at 33% recall will be the maximum precision among $9th$ to $20th$ Outputs:

$$\begin{aligned} \text{Interpolated precision @ } 33\% &= \max(\frac{3}{9}, \frac{3}{10}, \frac{4}{11}, \frac{4}{12}, \frac{4}{13}, \frac{4}{14}, \frac{5}{15}, \frac{5}{16}, \frac{5}{17}, \frac{5}{18}, \frac{5}{19}, \frac{6}{20}) \\ &= \frac{4}{11} = 36.36\% \end{aligned} \tag{8}$$

(5) What is the **MAP** for the query?
Since we have only one result set, the **MAP** is the *Average Precision* of this result set.

$$\begin{aligned} \mathbf{MAP} &= \frac{\frac{1}{1} + \frac{2}{2} + \frac{3}{9} + \frac{4}{11} + \frac{5}{15} + \frac{6}{20}}{8} \\ &= \frac{1099}{2640} = 41.63\% \end{aligned} \tag{9}$$

(6) What is the largest possible **MAP** that this system could have?
The earlier the relevant documents are retrieved, the larger the **MAP** will be.
We already have 6 relevant documents retrieved in $top-20$ returns, so the largest possible **MAP** is under the case where $21st$ and $22nd$ Outputs are relevant documents.

$$\begin{aligned}\textbf{MAP} &= \frac{\frac{1}{1}+\frac{2}{2}+\frac{3}{9}+\frac{4}{11}+\frac{5}{15}+\frac{6}{20}+\frac{7}{21}+\frac{8}{22}}{8} \\ &= \frac{443}{880} = 50.34\%\end{aligned} \tag{10}$$

(7) What is the smallest possible **MAP** that this system could have?
The later the relevant documents are retrieved, the smaller the **MAP** will be.
So the smallest possible **MAP** is under the case where $9999th$ and $10000th$ Outputs are relevant documents.

$$\begin{aligned}\textbf{MAP} &= \frac{\frac{1}{1}+\frac{2}{2}+\frac{3}{9}+\frac{4}{11}+\frac{5}{15}+\frac{6}{20}+\frac{7}{9999}+\frac{8}{10000}}{8} \\ &= 41.65\%\end{aligned} \tag{11}$$

(8) How large (in absolute terms) can the error for the **MAP** be?
The error between the approximated **MAP** and the smallest possible **MAP** is $41.65\% - 41.63\% = 0.02\%$.

The error between the approximated **MAP** and the largest possible **MAP** is $50.34\% - 41.63\% = 8.71\%$.

The error for the **MAP** by calculating (5) instead of (6) and (7) for this query can be ranges from 0.02% to 8.71%.

Q4. (1) Compute the likelihood of the query for both d_1 and d_2, without smoothing.

$$\begin{aligned}p(Q|d) &\approx p(Q|M_d) \\ &= \prod_{w\in Q} p(w|M_d) \\ &= \prod_{w\in Q} \frac{tf_{(w,d)}}{dl_d}\end{aligned} \tag{12}$$

So we can calculate $p(Q|d_1)$ and $p(Q|d_2)$ respectively:

$$\begin{aligned}p(Q|d_1) &= \prod_{w\in Q} \frac{tf_{(w,d_1)}}{dl_{d_1}} \\ &= \frac{2}{10}\times\frac{3}{10}\times\frac{1}{10}\times\frac{2}{10}\times\frac{2}{10}\times\frac{0}{10} \\ &= 0\end{aligned} \tag{13}$$

$$\begin{aligned}p(Q|d_2) &= \prod_{w\in Q} \frac{tf_{(w,d_2)}}{dl_{d_2}} \\ &= \frac{7}{10}\times\frac{1}{10}\times\frac{1}{10}\times\frac{1}{10}\times\frac{0}{10}\times\frac{0}{10} \\ &= 0\end{aligned} \tag{14}$$

Due to not adopting any smoothing method, as long as there is **any** word with **zero** occurrence in particular document, the whole **Query** with respect to that document has **0** probability.
So here $p(Q|d_1) = p(Q|d_2) = 0$, we can not ranked any document higher than another.

(2) Compute the likelihood of the query for both d_1 and d_2, do smooth with Jelinek-Mercer's method.
The Jelinek-Mercers model (with $\lambda = 0.8$):

$$p(w|d) = \lambda p(w|M_d) + (1-\lambda)p(w|M_c) \tag{15}$$

So the likelihood of query is calculated as:

$$\begin{aligned} p(Q|d) &= \prod_{w \in Q} p(w|d) \\ &= \prod_{w \in Q} \lambda p(w|M_d) + (1-\lambda)p(w|M_c) \end{aligned} \tag{16}$$

Now we can calculate $p(Q|d_1)$ and $p(Q|d_2)$ respectively:

$$\begin{aligned} p(Q|d_1) &= \prod_{w \in Q} \lambda p(w|M_{d_1)+(1-\lambda)p(w|M_c)} \\ &= (0.8 \times \frac{2}{10} + 0.2 \times 0.8) \times (0.8 \times \frac{3}{10} + 0.2 \times 0.1) \times (0.8 \times \frac{1}{10} + 0.2 \times 0.025) \\ &\times (0.8 \times \frac{2}{10} + 0.2 \times 0.025) \times (0.8 \times \frac{2}{10} + 0.2 \times 0.025) \times (0.8 \times \frac{0}{10} + 0.2 \times 0.025) \\ &= 0.000000962676 = 9.6 \times 10^{-7} \end{aligned} \tag{17}$$

$$\begin{aligned} p(Q|d_2) &= \prod_{w \in Q} \lambda p(w|M_{d_2)+(1-\lambda)p(w|C)} \\ &= (0.8 \times \frac{7}{10} + 0.2 \times 0.8) \times (0.8 \times \frac{1}{10} + 0.2 \times 0.1) \times (0.8 \times \frac{1}{10} + 0.2 \times 0.025) \\ &\times (0.8 \times \frac{1}{10} + 0.2 \times 0.025) \times (0.8 \times \frac{0}{10} + 0.2 \times 0.025) \times (0.8 \times \frac{0}{10} + 0.2 \times 0.025) \\ &= 0.000000013005 = 1.3 \times 10^{-8} \end{aligned} \tag{18}$$

$p(Q|d_1) > p(Q|d_2)$, so document 1 d_1 would be ranked higher than document 2 d_2.