SRUJAN.R.
BT18CSE041

IR - Assignment -1

1. Given: 300 documents

$\text{Relevant}(q) = \{A, B, C\} = 3$

System-1: N N A B N

$\text{Rel}_{sys1} = \{A, B\} = 2$

$$\therefore \text{MAP}_{sys1} = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} \text{Precision}(R_{jk})$$

$$= \frac{1}{2}\left(\frac{1}{3} + \frac{2}{4}\right) = \frac{1}{\cancel{2}} \times \frac{\overset{5}{\cancel{10}}}{3 \times 4} = \frac{5}{12}$$

$$\simeq 0.4167$$

System-2: N A N N B

$\text{Rel}_{sys2} = \{A, B\} = 2$

$$\therefore \text{MAP}_{sys2} = \frac{1}{2}\left(\frac{1}{2} + \frac{2}{5}\right) = \frac{1}{2} \times \frac{9}{2 \times 5} = \frac{9}{20}$$

$$= 0.45$$

System-3: A N N N N

$\text{Rel}_{sys3} = \{A\} = 1$

$$\therefore \text{MAP}_{sys3} = \frac{1}{1}\left(\frac{1}{1}\right) = 1$$

$$\therefore \text{MAP}_{sys3} > \text{MAP}_{sys2} > \text{MAP}_{sys1}$$

As system 3 has the highest Maximum A posteriori (MAP), it should be preferred. MAP is a useful estimate which gives us a point estimate of the query results in these systems.

2. Given: t_1 = VNIT, t_2 = topper, t_3 = Medal

$q_0 = (1,1,0)$

Relevant $\begin{cases} d_1 = (1,2,1) \\ d_2 = (3,2,1) \end{cases}$ Non relevant $\{ d_3 = (6,0,0)$

$\alpha = \beta = \gamma = 1$

According to Rocchio's update rule,

$$\Rightarrow \vec{q_m} = \alpha \vec{q_0} + \beta \frac{1}{|D_r|} \sum_{d_j \in D_r} \vec{d_j} - \gamma \frac{1}{|D_{nr}|} \sum_{d_j \in D_{nr}} \vec{d_j}$$

$$\Rightarrow \vec{q_m} = 1(1,1,0) + 1 \times \frac{1}{2} \times [(1,2,1) + (3,2,1)]$$

$$- 1 \times \frac{1}{1} [(6,0,0)]$$

$$= (1,1,0) + (2,2,1) - (6,0,0)$$

$$= (-3, 3, 1)$$

$$\therefore \vec{q_m} = (0,3,1)$$

All the documents given will have a lot of occurences of the term "VNIT", hence its weight will be irrelevant. The weights of "topper" and "medal" are more important. The initial query q_0 (1,1,0) didn't consider the weight of terms "Medal" and hence was wrong. The new vector now considers the weight of "medal" and "topper" which also satisfies relevant documents.

3. Given: Doc1: Phone ring person happy person
Doc2: Dog pet happy run jump
Doc3: Cat purr pet person happy
Doc4: Life smile run happy
Doc5: Life laugh walk run

(a) Term Document Frequency:

	D_1	D_2	D_3	D_4	D_5
cat	0	0	1	0	0
dog	0	1	0	0	0
happy	1	1	1	1	0
jump	0	1	0	0	0
laugh	0	0	0	0	1
life	0	0	0	1	1
person	1	0	1	0	0
pet	0	1	1	0	0
phone	1	0	0	0	0
purr	0	0	1	0	0
ring	1	0	0	0	0
run	0	1	0	1	1
smile	0	0	0	1	0
walk	0	0	0	0	1

(b) Inverted index for ranked retrieval

cat → [d_3] ⏚

dog → [d_2] ⏚

happy → [d_1] — [d_2] — [d_3] — [d_4] ⏚

jump → [d_2] ⏚

laugh → [d_5] ⏚

life → [d_4] — [d_5] ⏚

person → [d_1] — [d_3] ⏚

pet → [d_2] — [d_3] ⏚

phone → [d_1] ⏚

purr → [d_3] ⏚

ring → [d_1] ⏚

run → [d_2] — [d_4] — [d_5] ⏚

smile → [d_4] ⏚

walk → [d_5] ⏚

4. When handling a query containing multiple terms, some terms should be given more importance compared others and the scores of all terms of the query along with the scores of documents contained these terms should be computed accordingly.

- **Computing Cosine Scores:**
  For each query term, the weight of that term in the query is calculated and its posting list is fetched. Then for every <document, term frequency> pair, score of the document is updated to the previous score plus the product of weight of term in query and the (wf) of term in the document. The top k such documents are returned to the user.

- **Fast Cosine Scores:**
  If weight of all the terms in a query are considired equal, then instead of taking weights of term, it can be replaced with 1. Hence, only (wf) of the term in a document is added to the scores of a document for each term.

For top k document retrieval, max-heap can be used. Construction of heap will take O(N) and top k entries can be read in O(logN). Thus, complexity becomes O(klogN). For further optimization, we can avoid computing scores for all N documents and consider inexact k documents only, i.e, computing only for those k documents which are closer to k-best. Also, we can use Index Elimination, where we consider only those documents which have atleast one query term.

- Champion List:
  A set of r documents are precomputed by every term in dictionary. These r docs have highest term freq values. These r docs set is the champions list for the given terms. Now, for every given query, union of champion list for all terms in query can be considered for cosine computation which improves efficiency to a good scale.

- Cluster Pruning:
  Document vectors are clustered as a preprocessing step. Then at query time only that cluster is consider which is closer to the query. Randomly some leaders (docs) are selected and at query time, the cluster (leader + followers) closer to the query is considered for top k document retrieval

5. Basic principle of KMP algorithm is minimizing the manual matching which is done in the brute force method, having the worst complexity of O (pattern size * text size). This is done by keeping track of the

characters that are gonna match anyway once a mismatch is found, & in other words we precompute the length of the longest prefix which is also a suffix at every index of the pattern.

Ex: for the pattern "AAAA"
lps[] = [0, 1, 2, 3]

for the pattern "ABCDE"
lps[] = [0, 0, 0, 0, 0]

Now, when we slide the pattern one by one and compare the character, we use the value from lps[] to decide the next characters of the pattern to be matched. Therefore, not matching a character that we know will anway match.

Ex: txt = "AAAAA BAAABA", pat = "AA AA"
lps[] = [0, 1, 2, 3], i = txt pointer
j = pattern pointer

```
i=0, j=0
AAAAABAAABA
AAAA

i=1, j=1
AAAAABAAABA
AAAA

i=2, j=2
AAAAABAAABA
AAAA

i=3, j=3
AAAAABAAABA
AAAA

i=4, j=4
AAAAABAAABA
 AAAA
```

} here we are directly checking for only the last character of the pattern