# 1 Hyper-Parameter Tuning and Scores

## 1.1 Task 1

I analysed various choices and combinations of query terms (query, question, narrative) and mu (Smoothing Hyper-Parameter) and then found the following results :-

### 1.1.1 query

| $\mu$ | nDCG | nDCG@5 | nDCG@10 | nDCG@50 |
|---|---|---|---|---|
| 1000 | 0.1388 | 0.6152 | 0.5378 | 0.4899 |
| 200 | 0.1404 | 0.6174 | 0.5894 | 0.5121 |
| **100** | **0.1417** | 0.6303 | 0.6055 | 0.5193 |
| 50 | 0.1417 | 0.6252 | 0.6091 | 0.5197 |
| 10 | 0.1399 | 0.5991 | 0.5618 | 0.5082 |

### 1.1.2 question

| $\mu$ | nDCG | nDCG@5 | nDCG@10 | nDCG@50 |
|---|---|---|---|---|
| 200 | 0.1415 | 0.6028 | 0.6089 | 0.5139 |
| **100** | **0.1421** | 0.6249 | 0.6094 | 0.5159 |
| 50 | 0.1416 | 0.6300 | 0.5932 | 0.5122 |
| 20 | 0.1407 | 0.6026 | 0.5800 | 0.5056 |
| 10 | 0.1399 | 0.5745 | 0.5680 | 0.4965 |
| 1 | 0.1363 | 0.5207 | 0.5297 | 0.4700 |

### 1.1.3 narrative

| $\mu$ | nDCG | nDCG@5 | nDCG@10 | nDCG@50 |
|---|---|---|---|---|
| 200 | 0.1415 | 0.6028 | 0.6089 | 0.5139 |
| **100** | **0.1421** | 0.6249 | 0.6094 | 0.5159 |
| 50 | 0.1416 | 0.6300 | 0.5932 | 0.5122 |
| 10 | 0.1399 | 0.5745 | 0.5680 | 0.4965 |

Thus, Using the "question" field from the query file, along with $\mu$ = 100 seems to be the best option. In hindsight, the best results probably come from "question" because it depicts the information need (better than query) and is concise (unlike the narrative)

## 1.2 Task 2

From my analysis of the previous part, I had recognised that $\mu$ 50 or 100 were the best-performing values. Now, I analysed various choices and combinations of query terms (query, question, narrative), mu (50 or 100) and number of expanded words to be added (1, 2, 5, 10 or 20) and found the following results :-

### 1.2.1 query

| $\mu$ | #expansion words | nDCG | nDCG@5 | nDCG@10 | nDCG@50 |
|---|---|---|---|---|---|
| 50 | 20 | 0.1272 | 0.4253 | 0.4075 | 0.3865 |
| 50 | 10 | 0.1372 | 0.5729 | 0.5469 | 0.4913 |
| 50 | 5 | 0.1400 | 0.6029 | 0.5725 | 0.5056 |
| 50 | 2 | 0.1398 | 0.6089 | 0.5925 | 0.5123 |
| 50 | 1 | 0.1409 | 0.6224 | 0.6036 | 0.5191 |
| 100 | 20 | 0.1272 | 0.4253 | 0.4075 | 0.3865 |
| 100 | 10 | 0.1399 | 0.6014 | 0.5833 | 0.5010 |
| 100 | 5 | 0.1399 | 0.5964 | 0.5784 | 0.5057 |
| 100 | 2 | 0.1401 | 0.6104 | 0.5937 | 0.5123 |
| 100 | 1 | 0.1407 | 0.6268 | 0.5998 | 0.5160 |

### 1.2.2 question

| $\mu$ | #expansion words | nDCG | nDCG@5 | nDCG@10 | nDCG@50 |
|---|---|---|---|---|---|
| 50 | 10 | 0.1272 | 0.4253 | 0.4075 | 0.3865 |
| 50 | 5 | 0.1384 | 0.6207 | 0.5797 | 0.4886 |
| 50 | 2 | 0.1415 | 0.6370 | 0.5996 | 0.5150 |
| 100 | 10 | 0.1272 | 0.4253 | 0.4075 | 0.3865 |
| 100 | 5 | 0.1383 | 0.6066 | 0.5827 | 0.4873 |
| **100** | **2** | **0.1417** | 0.6271 | 0.6145 | 0.5166 |

### 1.2.3 narrative

| $\mu$ | #expanded words | nDCG | nDCG@5 | nDCG@10 | nDCG@50 |
|---|---|---|---|---|---|
| 100 | 10 | 0.1272 | 0.4253 | 0.4075 | 0.3865 |
| 100 | 5 | 0.1383 | 0.6066 | 0.5827 | 0.4873 |
| 100 | 2 | 0.1417 | 0.6271 | 0.6145 | 0.5166 |

Thus, Using the "question" field from the query file, along with $\mu = 100$ and number of expanded words = 2 seems to be the best option. Adding more words to the expanded list seems to decrease performance by shifting the context elsewhere.

# 2 Running Details and File Structure

## 2.1 Task 1 - RM1

1. Call **lm_rerank.sh** with the apt arguments. The arguments expected by this file (in order) are called as **bash lm_rerank.sh [query-file] [top-100-file] [collection-dir] [output-file]**

2. The script calls **top_rerank_rm1.py**, and passes the same arguments as above to it. This file is the flow-control of the entire task.

3. **top_rerank_rm1.py** calls a series of healper files for reading some of the data. These are :- * **read_csv.py** :- Houses functionality to read the ‘metadata.csv in the ‘[collection-dir]‘ * **read_qfile.py** :- Houses functionality to read and parse the ‘[query-file]‘ * **read_top100.py** :- Houses functionality to read and parse the ‘[top-100-file]‘ * **read_tjson.py** :- To read and parse a json file (pmc_json/pdf_json)

4. **rm1.py** is where the algorithmic implementations of the *RM1* model are housed. These include computing the per-document LM, the background LM, and functionality for calculating the query-document score. This score can now be used to re-rank the documents for a given query.

5. **top_rerank_rm1.py** iterates over the queries. For each query, it computes a score for the top100 documents that have been retrieved . Arranging these is a descending order, these results are written to the ‘[output-file]‘.

## 2.2 Task 2 - W2V

1. Call **w2v_rerank.sh** with the apt arguments. The arguments expected by this file (in order) are called as **bash w2v_rerank.sh [query-file] [top-100-file] [collection-dir] [output-file] [expansions-file]**

2. The script calls **top_rerank_w2v.py**, and passes the same arguments as above to it. This file is the flow-control of the entire task.

3. **top_rerank_w2v.py** calls a series of healper files for reading some of the data. These are :- * **read_csv.py** :- Houses functionality to read the ‘metadata.csv in the ‘[collection-dir]‘ * **read_qfile.py** :- Houses functionality to read and parse the ‘[query-file]‘ * **read_top100.py** :- Houses functionality to read and parse the ‘[top-100-file]‘ * **read_tjson.py** :- To read and parse a json file (pmc_json/pdf_json)

5. **word2vec** trains on the **intm_data/i.txt** for the ith query, and yields a **intm_data/vector_i.bin**, from which the embedding matrix U can be extracted for this query. This is then used to calculate a per-term score for terms in the vocabulary via $U * U^T * q$, and the top-k terms thus appearing are selected as expansion terms.

6. These expansion terms are appended to the original query terms, and Task - 1 is then applied for re-ranking the scores.

6. The re-ranked results are written to the [output-file] and the expansions to [expansions-file]

# 3 Algorithm and Implementation Details

## 3.1 Task 1

**Algorithm & Implementation of RMI**

1. Input bash lm_rerank.sh <input filenames>
2. top_rerank_rml.py — Orchestrates code execution
3. Read Queries from [query-file] using read_qfile.py
4. Read, parse & store the top 100 files from [top-100-file] corresponding to each query entry using read_top100.py
5. For each query, access pmc/pdf file location for each of it's top-100 retrieved files from metadata.csv, if available, using read_csv.py

- pmc file available → 6. read & parse pmc file using read_tjson.py
- pdf file available → If pmc also available, ignore pdf. If not, read & parse pdf using read_tjson.py
- both unavailable → make-do with fields extracted from metadata.csv.

7. For each of the 100 files, compute a local distbn. over the 100 files, compute a background distbn.

8. For every word $w$ in the vocabulary of this query,

$$\text{Score}[\text{document } d] \mathrel{+}= \underbrace{P(w|R)}_{\text{how relevant is this word}} \underbrace{\log(P(w|d))}_{\text{how often does it occur in the document}}$$

$$P(w|R) \sim P(w|q_1, q_2, q_3 \ldots q_k)$$
$$\propto P(w, q_1, q_2, q_3 \ldots q_k)$$
$$= \Sigma_M P(M)\, P(w|M) \prod_{i=1}^{k} p(q_i|M)$$

where $M_j$ would be the Model corresponding to $d_j$

& $p(w|M_j) = \dfrac{f_{w,d_j} + \mu \hat{P}_c(w)}{|d_j| + \mu}$ & $\hat{P}_c(w) = \dfrac{f_w}{|C|}$

collection (top 100 documents)

9. Rank the documents on ascending orders of their scores & print to the output file.

### 3.2 Task 2

W2V Implementation

For query i,

1. Create the training Corpuses for local embeddings by collecting text from the top 100 files for each query - store them in "intm_data/i.txt"

2. Train these .txt files using ./word2vec, and store the embedding matrices $U_i$ in "intm_data/vector_i.bin"

3. As referenced from the paper, Create a query vector $\mathbf{q}$, and Obtain distane scores using $UU^T q$, Sorting & thus picking top-k terms.

4. Append the expanded terms to the original query & Perform Task 1 reranking