COL764 Assignment 2 Report

Kshitij Alwadhi (2019EE10577)

12th October 2021

1 Introduction

In this assignment we develop telescoping models aimed at improving the precision of results using pseudo-relevance feedback. We will use data collection and queries from TREC COVID track. In the first part of the assignment, we implement Rocchio's method for re-ranking. In the second part of the assignment, we implement Language models based on Lavrenko and Croft's relevance models with a Unigram model with Dirichlet smoothing. At the end, we need to re-rank the top 100 results given to us.

2 Implementation Details

2.1 Pre-Processing the Data

Multiple steps are involved due to the structure of the data. One part of the data that we are given is the metadata.csv file which contains the document identifier and the paths to the respective documents. These documents are stored as json files which are opened as and when needed.

While reading the documents, we also perform stopword elimination and regex cleaning to remove unnecessary punctuation marks.

- **Parsing:** We first open the metadata.csv file. Each row corresponds to a document entry with our concerning columns being the title, abstract and the paths of the files given in the document parses folder. For each of the document that we are given as a json file, we extract the 'text' field from all the subsections in body text.
- **Regex Cleaning:** Next step is that we need to clean the text document from the unwanted punctuation marks. To do this, we use Regex pattern matching and split the text accordingly.
  The following delimiters are used: `[ ’,(){}.:;"‘\n]`.
- **Stopwords:** We also use the stopwords given by the NLTK library to remove words which are used very frequently and don't play any significant role from an Information Retrieval point of view.

2.2 Vector Space Model for Calculating Similarity

In vector space model, we represent our query and the documents as vectors in a t-dimensional space, where t is the size of our vocabulary. The document and the query can be represented as:

$$\vec{d_j} = (w_{1,j}, w_{2,j}, ..., w_{t,j})$$
$$\vec{q} = (w_{1,q}, w_{2,q}, ..., w_{t,q})$$

For computing the similarity, we define a degree of similarity of the document and a query as:

$$sim(d_j, q) = cos(\theta) = \frac{\vec{d_j}.\vec{q}}{|\vec{d_j}||\vec{q}|}$$

The intuition is that, the more similar the query and a document are, the higher rank the document is given in our top-k results. This similarity is indicated from the metric above.

The weights of the vector above are represented by the TF-IDF product. TF-IDF stands for "Term Frequency - Inverse Document Frequency". In this technique, we quantify the presence of a word in a documen, we generally compute a weight to each word which signifies the importance of the word in the document and our corpus. We define TF, IDF as follows:

$$tf_{i,j} = 1 + log(f_{i,j})$$

$$idf_i = log(1 + \frac{N}{df_i})$$

$$\text{TF-IDF Score} = tf_{i,j} * idf_i$$

2.3 Rocchio Re-rank

In the first part of the assignment, we are asked to implement Rocchio's method for re-ranking.

2.3.1 Query reformulation

Here, first we create the vectors corresponding to the queries given to us. Next, for the task of query reformulation, we need document vectors for relevant and non-relevant documents. Since we won't be having a direct document relevance information available to us, we resort to a different approach where we consider the documents given to us in the top-100 as relevant documents and a random set of 1000 documents as non-relevant for the particular query.

$$\vec{q_r} = \alpha * \vec{q} + \frac{\beta}{|R|} * \sum_{d \in R} \vec{d} - \frac{\gamma}{|NR|} * \sum_{d \in NR} \vec{d}$$

2.3.2 Re-ranking

Once we have the reformed query vectors, we calculate the similarity between the documents and the query using the similarity metric defined in the Vector Space Model. Next, we sort our documents on the basis of the similarity score with the document having the score being ranked 1. This way, for every query we have a re-ranked set of 100 documents with their associated scores.

2.3.3 Evaluation Schema

For evaluating the performance of the model for a particular set of α, β, γ, we calculate the NDCG @100 score of every query and take an average over the scores obtained for the queries. The higher this score is, the better our model is assumed to be performing. For calculating these NDCG values, we make use of the Qrels file given to us with the relevance values of a set of documents for a particular query. For the documents whose information is not given to us in the Qrels file but present in the Top-100 file, we assume their relevance to be 0 (Non-Relevant).

2.3.4 Hyper-parameter Tuning

For tuning the parameters, we perform a grid search over a range of values for α, β, γ.

```
def gridSearch():
    for alpha in [0.5, 0.75, 1.0, 1.25, 1.5, 1.75, 2.0]:
        for beta in [0.15, 0.25, 0.4, 0.5, 0.75, 1.0, 1.25]:
            for gamma in [0.05, 0.1, 0.15, 0.2]:
                updated_queries = rocchio_updateQueriesTest(alpha, beta, gamma)
                allscores, ndcg = rocchio_getNDCG(updated_queries)
                # Use the NDCG score generated above as a metric for comparing
                # the performance of different tuples.
```

α	β	γ	NDCG
2	0.15	0.2	0.725378
1.75	0.15	0.2	0.723731
2	0.15	0.15	0.722582
2	0.15	0.1	0.721097
1.75	0.15	0.15	0.720505
2	0.15	0.05	0.719051
1.5	0.15	0.2	0.718734
1.75	0.15	0.1	0.71845
1.25	0.15	0.2	0.718259

Table 1: Top 10 results from the grid search

Following are some of the graphs that we get of the variation of nDCG with change in α, β, γ.

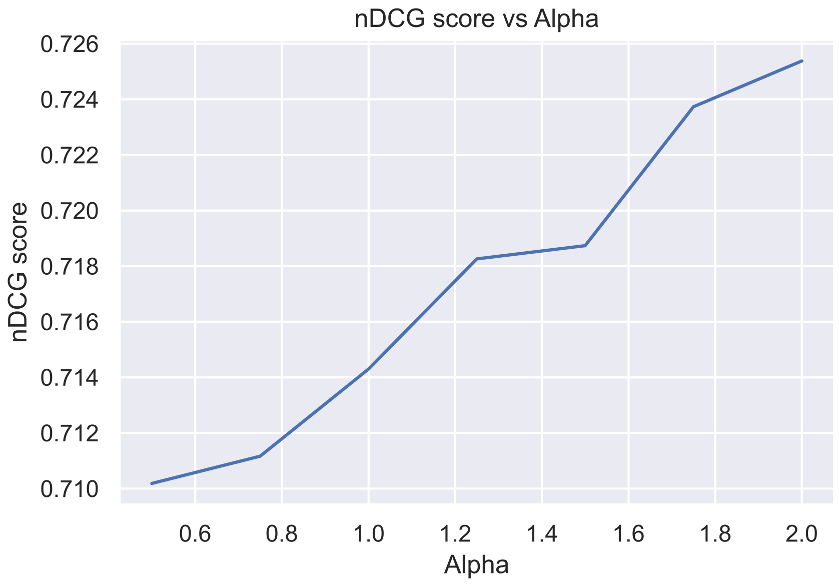

Figure 1: nDCG vs α

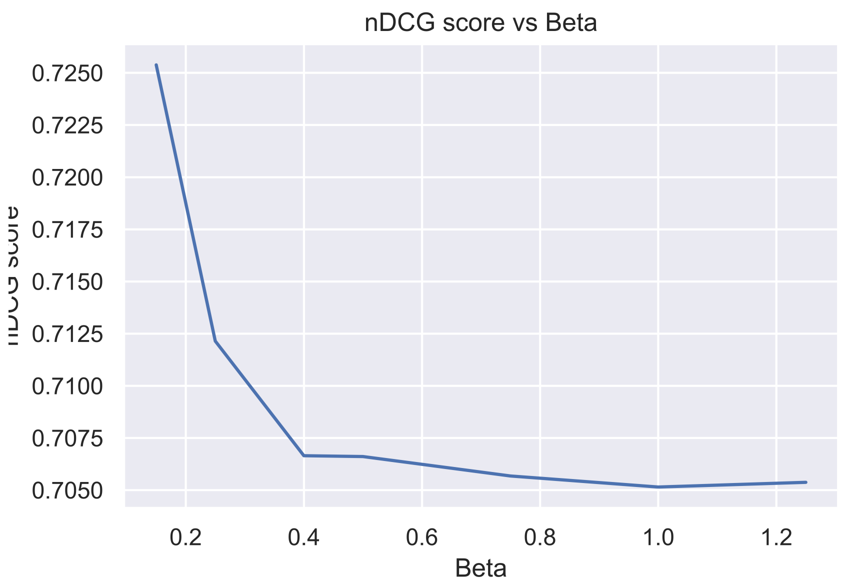

Figure 2: nDCG vs β

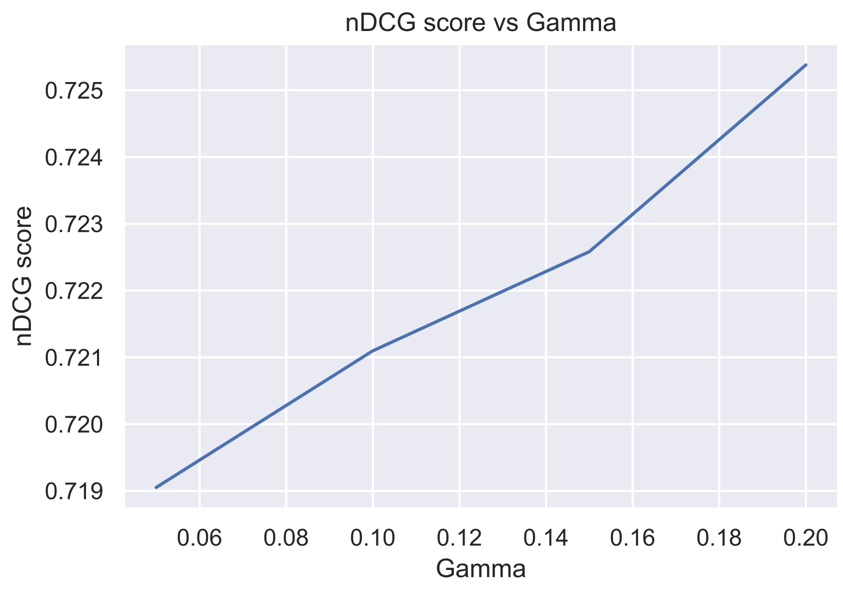

Figure 3: nDCG vs γ

Hence, from these experiments, we settle for the following hyper-parameters:

Hyper-parameter	Value
α	2.0
β	0.15
γ	0.2

Table 2: Hyper-parameters chosen for Rocchio Re-rank

2.4 Relevance Model based Language Modeling

In this part of the assignment, we use Lavrenko and Croft's relevance models with an Unigram model with Dirichlet smoothing. We implement both the RM1 and RM2 model for incorporating relevance information into the language models. In RM1 we use i.i.d. sampling and in RM2 we use conditional sampling. We also tune the μ parameter for Dirichlet smoothing.

2.4.1 Query Expansion

We represent the terms of the query as q_i and the probability of a word being considered for query expansion is given as:

$$P\left(w, q_1 \ldots q_k\right)=P(w) \prod_{i=1}^k P\left(q_i \mid w\right)$$

As per the paper by Lavrenko and Croft, there are two approaches that can be followed for calculating this probibility.

1. **I.I.D. Sampling:** Let's assume that the query words q1....qk and the words w in relevant documents are sampled identically and independently from a unigram distribution C, the collection of documents given to us. Let M_R represent some finite universe of unigram distributions from which we could sample. The sampling process proceeds as follows: we pick a distribution M $\in M_R$ with probability P(M) and sample from it k+1 times. Then the total probability of observing w together with q1....qk is

$$P\left(w, q_1 \ldots q_k\right)=\sum_{M \in \mathcal{M}} P(M) P(w \mid M) \prod_{i=1}^k P\left(q_i \mid M\right)$$

Here we assume that w and q1...qk are independent once we pick a distribution M.

2. **Conditional Sampling:** It is informative to contrast the assumptions made in the two methods we proposed for estimating the joint P(w,q1.....qk). By Bayes' rule, Method 1 can also be viewed as sampling of q1.....qk conditioned on w. However, for Method 1, this would add an additional constraint that all query words qi are sampled from the same distribution M, whereas in Method 2 we are free to pick a separate Mi for every qi. We believe that Method 1 makes a stronger mutual independence assumption, compared to a series of pairwise independence assumptions made by Method 2. In empirical evaluations, we found that Method 2 is less sensitive to the choice of our universe of distributions M. Method 2 also performs slightly better in terms of retrieval effectiveness and tracking errors.

$$\mathrm{P}(q_i \mid w)=\sum_{M_i \in \mathcal{M}} P\left(M_i \mid w\right) P\left(q_i \mid M_i\right) \; P\left(w, q_1 \ldots q_k\right)=P(w) \prod \sum_{M_i \in \mathcal{M}} P\left(M_i \mid w\right) P\left(q_i \mid M_i\right)$$

$$P\left(w, q_1 \ldots q_k\right)=P(w) \prod_{i=1}^k \sum_{M_i \in \mathcal{M}} P\left(M_i \mid w\right) P\left(q_i \mid M_i\right) \; P(w)=\sum_{M \in \mathcal{M}} P(w \mid M) P(M)$$

$$P\left(M_i \mid w\right)=\frac{P\left(w \mid M_i\right) P\left(M_i\right)}{P(w)}$$

We also make the following approximations for calculating the probabilities:

1. $P_C(t)=\frac{tf(t)}{100}$ where tf(t) = term frequency of the term t.
2. $P(M_i)=\frac{1}{100}$.
3. P(q1,q2..qk) can be ignored as they are constant and doesn't affect relative ranking of documents.
4. We consider the vocabulary to be the words present in the top100 documents given to us after tokenization.

A mathematical overview of the formulae is also attached at the end of this pdf.

After calculating these probabilities, we pick the top 20 words (sorted according to decreasing probabilities) and add them to our query. An example of the words we get after this query expansion are:

Original query: coronavirus response to weather changes

Expansion words: virus, data, temperature, weather, cases, may, also, health, climate, number, transmission, time, population, study, humidity, disease, change, new, countries, infection.

2.4.2 Re-Ranking

Once we have the expanded query vectors, we calculate the similarity between the documents and the query using the similarity metric defined in the Vector Space Model. Next, we sort our documents on the basis of the similarity score with the document having the score being ranked 1. This way, for every query we have a re-ranked set of 100 documents with their associated scores.

2.4.3 Evaluation Schema

We follow a similar approach as in the case of Rocchio re-ranking. For evaluating the performance of the model for a value of μ, we calculate the NDCG @100 score of every query and take an average over the scores obtained for the queries. The higher this score is, the better our model is assumed to be performing. For calculating these NDCG values, we make use of the Qrels file given to us with the relevance values of a set of documents for a particular query. For the documents whose information is not given to us in the Qrels file but present in the Top-100 file, we assume their relevance to be 0 (Non-Relevant).

2.4.4 Hyper-parameter Tuning

The only parameter we have in this case is the μ parameter, also known as the Dirichlet parameter. We perform a grid search for tuning this parameter as well.

```
def gridSearch():
    gridvals = [1e-12, 1e-10, 1e-8, 1e-6, 1e-4, 1e-2, 1, 1e2,\
    1e4, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
    for mu in gridvals:
        updated_queries_1 = LM_updateQueries(1, topK, mu)
        allscores_1, ndcg_1 = LM_getNDCG(updated_queries_1)
        updated_queries_2 = LM_updateQueries(2, topK, mu)
        allscores_2, ndcg_2 = LM_getNDCG(updated_queries_2)
        # Use these scores for analysis
```

Finally, we get the following results:

mu	nDCG_R1	nDCG_R2
1.00E-12	0.7281	0.732854
1.00E-10	0.7281	0.732854
1.00E-08	0.7281	0.732854
1.00E-06	0.7281	0.732854
0.0001	0.727856	0.732905
0.01	0.732814	0.734307
1	0.737852	0.735136
100	0.737712	0.735313
10000	0.733127	0.731147
0.1	0.735652	0.736184
0.2	0.731719	0.735782
0.3	0.73191	0.735292
0.4	0.73206	0.735312
0.5	0.733741	0.73531
0.6	0.737891	0.735265
0.7	0.737852	0.735265
0.8	0.737852	0.735136
0.9	0.737852	0.735136

Table 3: Results from the grid search

Following are some of the graphs that we get of the variation of nDCG with change in μ:

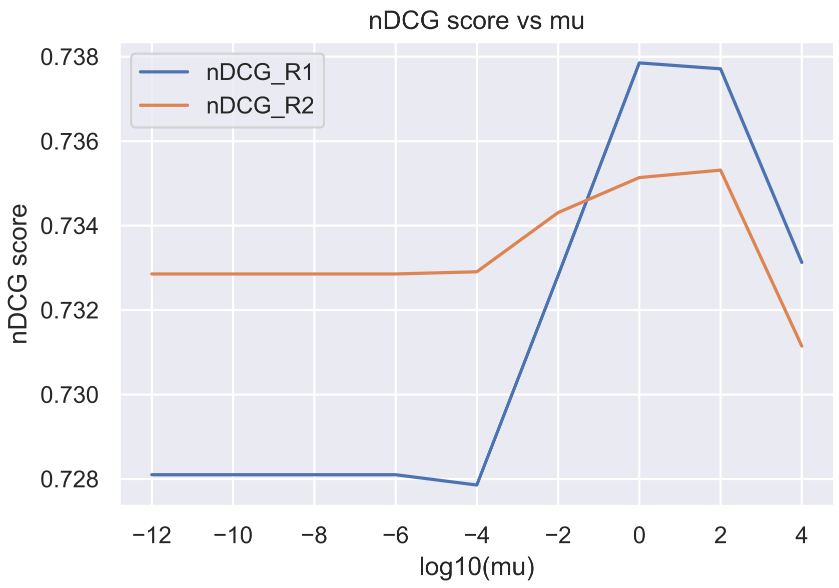

Figure 4: nDCG vs μ

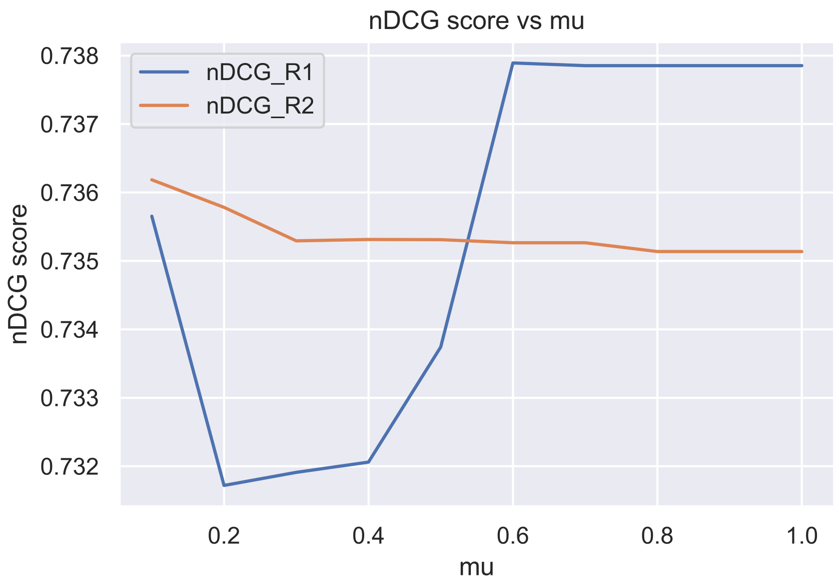

Figure 5: nDCG vs μ

Hence, from these experiments, we settle for the following value of μ:

Hyper-parameter	Value
mu	100

Table 4: Hyper-parameter chosen for Language Modelling

3 Final results using TREC-eval

The following are the results we get after running TREC-Eval:

	Original	**Rocchio**	**RM1**	**RM2**
nDCG @ 5	0.4844	0.5420	0.6046	0.6228
nDCG @ 10	0.4833	0.5255	0.5736	0.5897
nDCG @ 30	0.4613	0.4908	0.5270	0.5271
nDCG @ 100	0.4042	0.4130	0.4217	0.4231
MAP	0.0576	0.0584	0.0599	0.0602

Table 5: TREC eval results for different re-ranking methods

We can clearly see an improvement in all the metrics when we move from Rocchio to RM1 of Lavrenco Croft model and then to RM2 of Lavrenco Croft model.

Our Final Parameters are:

Hyper-parameter	**Value**
α	2.0
β	0.15
γ	0.2
μ	100

Table 6: Hyper-parameters chosen for Rocchio Re-rank

4 References

1. Victor Lavrenko and W. Bruce Croft. 2001. Relevance based language models. In Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '01). Association for Computing Machinery, New York, NY, USA, 120.

Language Modelling formulae

Kshitij Alwadhi
2019EE10577

For ranking,

$$\frac{P(D|R)}{P(D|N)} = \prod_{w \in D} \frac{P(w|R)}{P(w|N)}$$

where,

$$P(w|R) \simeq P(w|q_1 \dots q_k)$$

$$P(w|R) \simeq \frac{P(w, q_1 \dots q_k)}{P(q_1 \dots q_k)}$$

Method 1 :

$$P(w, q_1 \dots q_k) = \sum_{M \in \mathcal{M}} P(M) \cdot P(w|M) \prod_{i=1}^{k} P(q_i|M)$$

Method 2 :

$$P(w, q_1 \dots q_k) = P(w) \cdot \prod_{i=1}^{k} \sum_{M_i \in \mathcal{M}} \underbrace{P(M_i|w)}_{} \cdot P(q_i|M_i)$$

$$P(M_i|w) = \frac{P(w|M_i) \cdot P(M_i)}{P(w)}$$

Approximations

$$P(q_1 \dots q_k) = \sum_{w} P(w, q_1 \dots q_k)$$

$$P(w) = \sum_{M \in \mathcal{M}} P(w|M) \cdot P(M)$$

$$P(t|M_D) = \frac{f_{t,D} + \mu \boxed{P_c(t)}}{|D_j| + \mu} \qquad P_c(t) \rightarrow \frac{f_{t,c}}{|c|}$$

$\mu \rightarrow$ to be tuned

$$P_c(t) = \frac{tf(t)}{100}$$

$P(m_i) = 1/100 \longrightarrow$ also constant.

$P(q_1 \ldots q_k) =$ ignore

$V \rightarrow$ vocab of 100 docs.

$$P(w \mid m_d) = \frac{tf(w,d) + \mu \cdot \frac{tf(w)}{100}}{|d_j| + \mu}$$

Final forms:

Method 1 :

$$P(w \mid R) \simeq \sum_{m \in \mathcal{M}} P(m) \cdot P(w \mid m) \cdot \prod_{i=1}^{k} P(q_i \mid m)$$

$$= C \cdot \sum_{m \in \mathcal{M}} P(w \mid m) \cdot \prod_{i=1}^{k} P(q_i \mid m)$$

$C \rightarrow$ const.

Method 2 :

$$P(w \mid R) \simeq \cancel{P(w)} \cdot \prod \sum_{m_j \in \mathcal{M}} \frac{P(w \mid m_j) \cdot P(m_j)}{\cancel{P(w)}} \cdot P(q_i \mid m_j)$$

$$= C \cdot \prod_{i=1}^{k} \sum_{m_j \in \mathcal{M}} P(w \mid m_j) \cdot P(q_i \mid m_j)$$

$C \rightarrow$ const.