COL 764 Assignment 2 - Document Reranking Task

2024 - Version 1.0

Version History

Revision	Date	Author(s)	Description
1.1	24 Sep 2024	Instructor	Corrected input format.
1.0	10 Sep 2024	Instructor	Initial Version.

Deadline

Deadline for final submission of the complete implementation, and the report on the algorithms as well as performance tuning: September 25th 2024, 09:00 AM.

Weightage

This assignment is evaluated against 90 marks. The tentative breakup of marks is given in the end of the document.

Instructions

1. This programming assignment is to be done by each student individually. Do not collaborate -either by sharing code, algorithm, and any other pertinent details- with each other.
2. All programs have to be written either using Python/Java/C/C++ programming languages only. Anything else requires explicit prior permission from the instructor. The machine we use to evaluate your submissions has the following versions:
   - **C**, **C++**: gcc12.3;
   - **Java**: Java (SE) 22 (we will not use OpenJDK);
   - **Python:** version 3.12.

   We recommend you to use same versions to avoid the use of deprecated/unsupported functions or take care to ensure required compatibility.
3. A single tar/zip of the source code has to be submitted. The zip/tar file should be structured such that
   - upon deflating all submission files should be under a directory with the student's registration number. E.g., if a student's registration number is `20XXCSXX999` then the zip submission should be named 20XXCSXX999.zip and upon deflating **all contained files** should be under a directory named ./20XXCSXX999 only (names should be in uppercase). Your submission might be rejected and not be evaluated if you do not adhere to these specifications.
   - apart from source files, the submission zip file should contain a build mechanism *if needed* (allowed build systems are Maven and Ant for Java, Makefile for C/C++). It is the responsibility of each student to ensure that it compiles and generates the necessary executable as specified. **Please include an empty file in case building is not required** *(e.g. if you are using Python)*. **Note that we will use only Ubuntu Linux machines to build and run your assignments.** So take care that your file names, paths, argument handling etc. are compatible.

Filename	Size	Records	Description
docs.tsv.gz	7.9GB	3,213,835	**tsv**: doc_id, url, title, body
queries.tsv.gz	637 Bytes	24	**tsv**: query_id, text
top100docs.tsv.gz	19,281 Bytes	2,400	**tsv**: query_id, doc_id, score
qrels.tsv.gz	22,346 Bytes	9,098	**tsv**: query_id, doc_id, relevance, iteration
word2vec.300d.txt.gz	295MB	3,02,867	contains, in each line, a term followed by its 300 dimensional dense vector of real values representing pretrained Word2Vec embeddings
glove.6B.300d.txt.gz	377MB	4,00,001	contains, in each line, a term followed by its 300 dimensional dense vector of real values representing pretrained GloVe embeddings

Table 2: Details of the files released for this assignment. All Gzipped files need to be unzipped before using them – they are compressed to reduce the space consumption. All files are made available in the following directory on HPC: `/scratch/cse/faculty/srikanta/COL764-A2-2024`

4. You *should not* submit data files, index files, dictionaries etc. If you are planning to use any other "special" library, please talk to the instructor first (or post on Teams).
5. **Note that there will be no deadline extensions. Apart from the usual "please start early" advise, I must warn you that this assignment requires significant amount of implementation effort, as well as some 'manual' tuning of parameters to get good speed up and performance. Do not wait till the end.**

1 Assignment Description

In this assignment the goal is to evaluate different models of pseudo-relevance feedback based query expansion and reranking, aimed at improving the precision and recall of results. We will use a well-known TREC benchmark for this purpose. All the required data is available from

1.1 Data Description

Document Collection: There are $3,213,835$ unique documents in the collection given to you in the form of a tab-separated file consisting of 4 fields: `doc_id`, `url`, `title` and `body`.

Queries: We will consider 45 topics from the TREC track. Each topic is indexed by a `query_id` contains a keyword `text` that is to be treated as the initial query fired by the user.

Pseudo-relevance Collection: Unlike previous assignment where you were supposed to build an index and a retrieval system, in this task you are only required to *rerank the top-100 documents* that are already retrieved through an initial model from the collection. **For each query, you will be provided upto 100 top-ranked documents that were retrieved.**

No Inverted Indexes Please!!

In this assignment we are not looking for an implementation of index or an end-to-end retrieval system. You only need to submit your implementation of "reranking" of the given top-100 results for each query. For this purpose, should you need to process the document collection (to collect some statistics), then you can compute it using any method and use it.

2 Task

Starting from the top-100 documents for each query, you are supposed to implement the following methods for reranking them to get higher-relevant documents in the higher ranks.

How did we get these top-100?

It is immaterial how these top-100 candidates were generated (i.e., which specific retrieval model was used). Also, you are free to use any form of document preprocessing – i.e., tokenization, lemmatization, stop-word elimination, entity recognition, etc. (if needed) – as long as they provide better retrieval quality.

2.1 Task 0: Relevance Model for Retrieval

As the underlying retrieval model, we will use the standard language modeling based retrieval model with Dirichlet smoothing. You are free to tune the μ hyper-parameter that controls the smoothing (and any other parameter that you may encounter) to achieve higher retrieval quality. It is *mandatory to report all parameter choices* made in your implementation. Note that you are not expected to implement the retrieval model that works on the entire corpus (so no need for full-fledged inverted indexing). The corpus is provided only for assistance in (a) getting actual document content from the document identifier in the top-100 documents file, (b) to estimate the term weight and other statistics that you may need for implementing the language modeling retrieval model. You are only required to implement the relevance model required for *reranking* the top-100 documents that are provided for each query. **You may assume that the `query-file` contains only those queries whose top-100 documents are available in the `top-100-file`** (see Section 2).

NOTE: This is a task-0 of the assignment because we are not looking for an explicit submission file for this model implementation. You will of course need to implement it, submit it (could be part of the implementation for remaining tasks), and there is a separate evaluation weightage associated with this.

2.2 Task 1: Use of Local Embeddings for Query Expansion

Embedding vocabulary terms as dense, real valued vectors in a low dimensional space has been shown to successfully encapsulate semantic concepts associated with them [Mikolov et al., 2013]. At their core, these embeddings allow one to use (dense)vector inner-products to find other words that are considered to be "close" to each other in semantics. Semantic relationships captured by such word embeddings have been used to improve the performance of various information retrieval tasks such as document ranking, query reformulation, relevance feedback , and end-to-end deep neural ranking models.

In [Diaz et al., 2016], authors introduced the idea of using *local embeddings* (i.e., locally computed word embeddings) to expand the query and then do the reranking using these embeddings. Specifically, their approach can summarized as follows:

1. Train Word2Vec embeddings based on the initially retrieved top-k documents (the pseudo-relevance set).
2. Use these embeddings to compute the nearest words to the original query terms, and use top-N expansion terms from that set.
3. Rerank the documents using the expanded query and present the results.

You can use the word2vec implementation from `https://github.com/dav/word2vec`. Please refer to the original paper for more details of the model and implementation details. Note that there are two ways to train word2vec – SkipGram and CBOW. You are free to use either one (as long as it improves the performance ;-)) **You are limited to using not more than 20 expansion terms (you can use less, but not more).**

2.3 Task 2: Use of Generic Embeddings for Query Expansion

Alternatively, one could simply consider the possibility of finding top-m nearest words to the original query terms from a **pretrained** word embedding model, and using them to find top-N expansion terms. For this task, you can use the following pretrained word2vec and GloVe embeddings:

- **Word2Vec:** We will use 300 dimensional pre-trained Word2Vec embeddings released by Google. These embeddings were trained on a Google News corpus.
- **GloVe:** GloVe [Pennington et al., 2014] are an improvement over Word2Vec model. We make use of their pretrained 300 dimensional embeddings that were trained on Wikipedia 2014 + Gigaword 5 corpus. Note that their vocabulary is uncased.

We will make both of these available in the same directory as other files of the assignment, so you don't need to download them repeatedly. **You are limited to using not more than 20 expansion terms (you can use less, but not more).**

Given the size of pretrained embeddings, it is unlikely that you will require any special assistance (in terms of other libraries) for efficiently computing the top-N expansion terms. You can make use of GenSim Python library.

2.4 Evaluation Metrics

We have shared 24 queries and corresponding top-100 documents and qrels to help you tune parameters. As you can see in these qrels, all queries are judged on 3-point relevance grade: (0: non-relevant, 1: partially relevant, and 2: relevant). We will use **nDCG@**$\{5, 10, 50\}$ for evaluation.

2.5 Program Structure

You are expected to submit two implementations:

1. Word2Vec Local Model –
   `w2v-local_rerank.sh [query-file] [top-100-file] [collection-file] [output-file] [expansions-file]`
2. Word2Vec Generic Model –
   `w2v-gen_rerank.sh [query-file] [top-100-file] [collection-file] [w2v-embeddings-file] [output-file] [expansions-file]`
3. GloVe Generic Model –
   `glove-gen_rerank.sh [query-file] [top-100-file] [collection-file] [glove-embeddings-file] [output-file] [expansions-file]`

where,

Scripts	
`w2v-local_rerank.sh`:	shell script file
`w2v-gen_rerank.sh`:	shell script file
`glove-gen_rerank.sh`:	shell script file
INPUT files	
`query-file`:	file containing the queries in the TSV form as in `queries.tsv`.
`top-100-file`:	full-path of the file containing the top100 documents (`top100docs.tsv`), which need to be reranked. As mentioned earlier, you can assume that only the queries that are in the `query-file` have their corresponding top-100 documents listed here.
`collection-file`:	full-path of the file containing the full document collection in TSV format.
`[w2v|glove]-embeddings-file`:	full-path of the file containing the word2vec or glove embeddings.
OUTPUT files	
`output-file`:	file to which the output in the trec_eval format has to be written
`expansions-file`:	specifies the file to which the expansion terms used for each query should be output.

2.6 Output formats

We will make use of trec_eval tool for generating nDCG scores. Therefore, it is important that you follow the exact format specification as required by the tool. To reiterate – white space is used to separate columns. The width of the columns in the format is not important, but it is important to have exactly six columns per line with at least one space between the columns.

2.6.1 Format of Output-File:

Lines of results are of the following form

```
1 Q0 pid1    1 2.73 runid1
1 Q0 pid2    2 2.71 runid1
1 Q0 pid3    3 2.61 runid1
1 Q0 pid4    4 2.05 runid1
1 Q0 pid5    5 1.89 runid1
```

where:

- the first column is the query id.
- the second column is currently unused and should always be "Q0".
- the third column is the identifier of the retrieved document in context of document ranking task.
- the fourth column is the rank at which the document is retrieved.
- the fifth column shows the score (integer or floating point) that generated the ranking. This score must be in descending (non-increasing) order.
- The sixth column is the ID of the run you are submitting (you can leave it as "runid1" – but **DO NOT** leave it empty)

2.6.2 Output format for expansions-file

We will use the following simple format to output the expansion terms used for each query:
`<query_id> : <expansion term1>, ... <expansion term20>`
The output file specified in `expansions-file` will consist of one such single line for each topic.

2.7 Submission Plan

All your submissions should strictly adhere to the formatting requirements given above.

- Submission of your source code on 25th September, and the final version of results.
- You should also submit a README for running your code, and a PDF document containing the implementation details as well as the resulting nDCG scores with various parameters that have been specified (preferably as a table).
  Name them README.{txt|md} **and** 20XXCSXX999.pdf **respectively**.
- The set of commands for evaluation of your submission roughly follows -

  Tentative Evaluation Steps (which will be used by us)

```
$ unzip 20XXCSXX999.zip
$ cd 20XXCSXX999
$ bash build.sh
$ bash w2v-local_rerank.sh <arguments>
$ bash w2v-gen_rerank.sh <arguments>
$ bash glove-gen_rerank.sh <arguments>
```

- Here is the link to TREC EVAL tool. `https://trec.nist.gov/trec_eval/`
- Only `numpy/scipy`, `nltk`, `gensim` and libraries distributed with Python (e.g. `re`) will be available in the Python evaluation environment. You can also assume that NLTK libraries that are available through `''nltk.download('popular')''` are already downloaded. Include the source code of any imports in your submission for other languages (after getting prior permission from the instructor). No external downloads will be allowed for any language environment at runtime – i.e., there may not be internet connectivity during runtime, so tread carefully. If in doubt, please post it on the Teams channel.

2.8 Tentative breakup of marks assignment

In general, a submission qualifies for evaluation if and only if it adheres to the specifications given above (arguments, structure, use of external libraries, correct output format, input format adherence, etc.). Given this requirement, the marks assignment for correct implementation of:

Base LM ranking	10
Local Word2Vec reranking	25
Generic Word2Vec reranking	15
Generic Glove reranking	15
README and algorithmic details documentation	10
Report	15
Total	90

References

[Diaz et al., 2016] Diaz, F., Mitra, B., and Craswell, N. (2016). Query expansion with locally-trained word embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.

[Mikolov et al., 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Proc. NIPS*.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.