Query Optimization using prior topical filtering

Akshat Khare, Srijan Sinha

Abstract: In general large document collections are stored on distributed platforms where every query is evaluated parallely across all the clusters and then the results are merged together. In these settings the resource usage can be reduced if the documents are clustered such that a query only needs to run on a few of the clusters and not all of them. For this it is essential that similar documents are stored together. We want to evaluate the feasibility of this idea by judging the fall in quality of retrieved results when it is run over only a selected subset of the document corpus. In the process we aim to improve querying speed when the retrieval is done on only a subset rather than the whole corpus (assuming retrieval over whole corpus is serialized and not parallel as in the distributed setting case)
The goal can be summarized as :
(1) Use only a subset of document corpus to generate results with minimal loss in quality (candidate set).
(2) Improve the querying speed due to lesser computation.

1. Introduction

1.1. Motivation

The motivation behind the process was of topical sharding and extracting results from subset of shards and using that instead of computation on all of them.

1.2. Problem Statement

We wish to extract relevant documents from a subset of universal document set without computing ranking over whole document set. We have precomputed document embeddings and in the real time we calculate the query embedding. We take a subset of the documents based on cosine similarity and see if we can get similar relevance results when our procedure is augmented with improved query speed.

2. Related Work

Kulkarni gave the results of sharding in her paper[1] in SIGIR2010: Topic based partitions for efficient and effective selective search. Here the authors have aimed at merging results across various storage nodes where the documents to the nodes are allocated using three methods :
(1) Random Allocation
(2) Source based Allocation
(3) Topic-based Allocation
But they finally use results from all the nodes to create the final retrieval ranking. We want to limit ourselves to only the 'relevant' clusters of documents.

3. Model and Methodology

3.1. Retrieval Algorithm

We took an existing retrieval algorithm. We augmented our algorithm atop it and saw the difference in performance metrics and speedup. We implemented a basic BM25 and modelled our procedure to perform better than the same.

3.2. Task Formulation

The whole process is as follows:

1. **BM25 Core**: Given TIPSTER data in assignment 1 we implement the Okapi BM25 in python.
2. **Document embedding**: Given training data, precompute document embeddings using GenSim's word2vec library. Training time should be reasonably low and embeddings should be computed fast.
3. **Query Document similarity in real time**: Given the queries, compute the query-document similarity in real time and give a preconfigured subset of documents on which BM25 is supposed to operate. Find the query embeddings and calculate cosine similarity.
4. **Compare with BM25 Core** Compare the results with BM25 Core and tabulate results.

4. Dataset Description

We took Disk 1 of TIPSTER data collection. Queries were of topic 51-100. The files were taken as it is from the scratch directory on hpc as in Assignment 1.

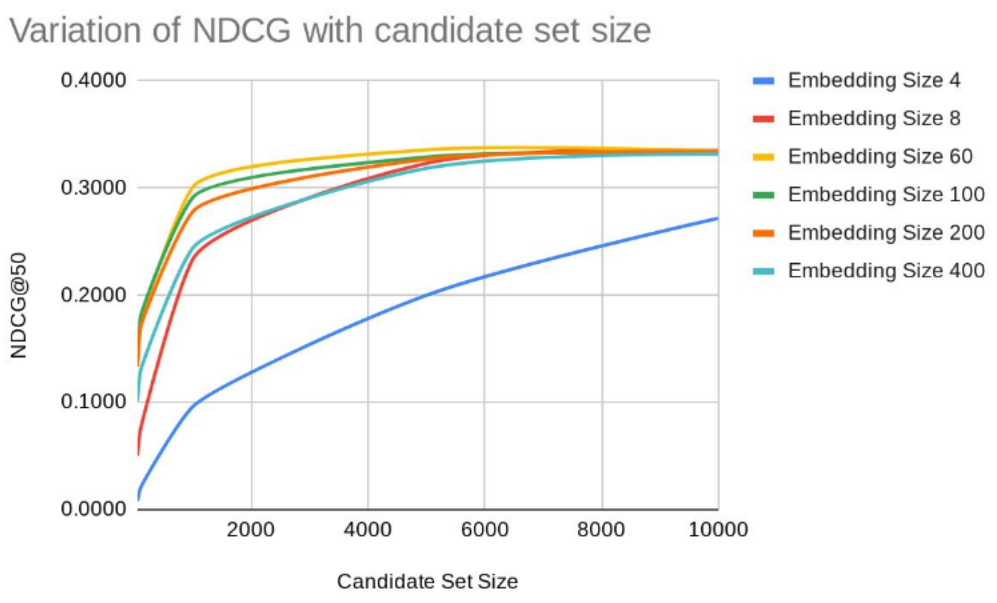
5. Experiment Setup

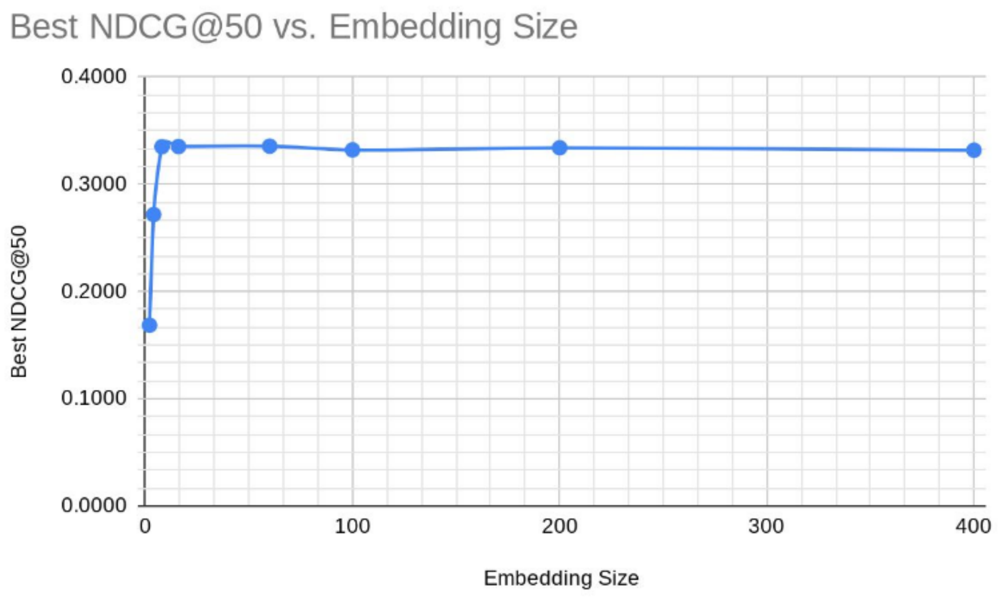
We used HPC machines with 2 cpu. Program was in python. Multiple runs were implemented to keep a warm cache. End to end time from reading queries from disk to calculating sorted results was tracked to avoid any unfair comparison. For the optimized code the query time included the time taken for converting queries to embedding and then finding the candidate set of documents for the BM25 Core algorithm.

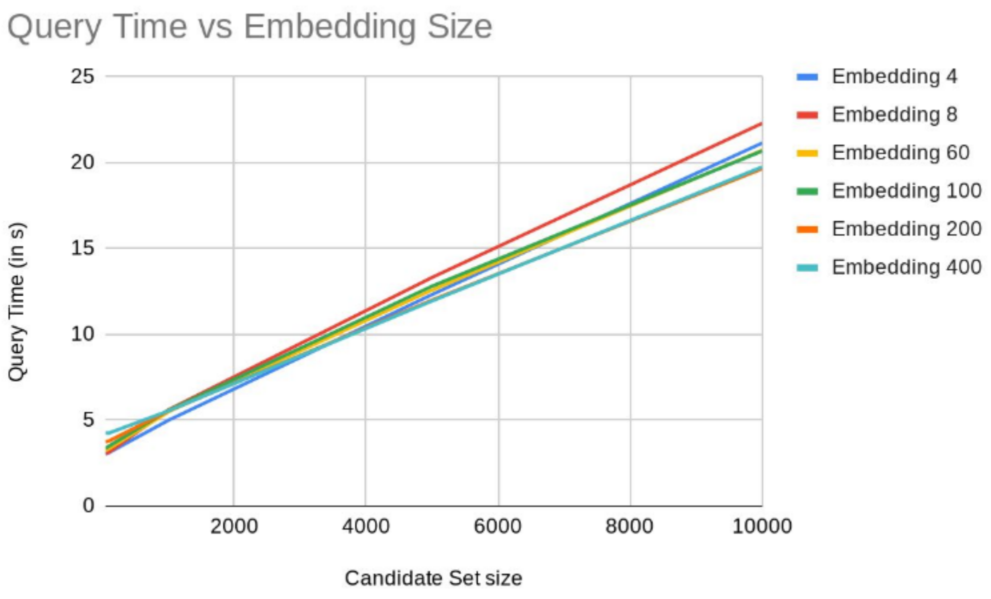
6. Results

6.1. Data

The results for the NDCG values and query runtimes of the optimized and the BM25 Core algorithm is as follows:







Candidate Set Size	Generation time over whole corpus and all queries
50	188 ms
100	237 ms
1000	1.91 s
5000	7.23 s
10000	19.65 s

The retrieval time for BM25 Core over whole corpus is nearly 90 seconds.
And the corresponding NDCG@50 is 0.3366.

6.2. Analysis

For higher embedding sizes in the range of 100-400 the algorithm manages to nearly touch the BM25 core NDCG with candidate set sizes of 1000 to 10000. If we calculate for a candidate set size of 1000 then:

$$t_c(50) = 1.91s$$

$$t_c(1) = 1.91/50 = 0.038s$$

Assuming that the time taken by BM25 varies linearly with the number of documents it covers. Then the ratio of run time of BM25 Core over the whole corpus to the run time of BM25 Core over the candidate set is given by:

$$t_{whole} : t_{candidate} = 80000 : 1000 = 80 : 1$$

So now if any BM25 implementation runs in t_{bm25} seconds for a query then for our algorithm to outperform this implementation in query speed the following equation must be satisfied:

$$t_{bm25} > t_c(1) + t_{bm25}/80$$

which on substituting $t_c(1)$ as above gives $t_{bm25} > 0.0389$ seconds

Any BM25 which runs in greater than this threshold value can benefit form using the topical selective search by sacrificing very little in quality for this corpus.

For ex. Jimmy Lin's Anserini tool which works on Lucene 7.6 gives a query time of 3.660 s over 50 queries which effectively evaluates to 732 ms per query. So we can say that since it is greater than the theoretical bound, our selective search will help in improving its speed with hopefully minimal reduction in quality.

The gain in speed that happens is due to the fact that while traversing the postings list we skip a few documents by checking if they are in the candidate set. This saves us a lot of arithmetic computation cycles.

7. Learnings

We learnt that BERT isn't always the answer to everything, for example here word2vec embeddings were much more light and effective than the BERT embeddings that we tried. We also learnt that arithmetic cost can't always be ignored and can lead to huge gains in the speed.

References

[1] Anagha Kulkarni, Jamie Callan, *Topic based Index Partitions for Efficient and Effective Selective Search.* ACM SIGIR 2010, https://dl.acm.org/citation.cfm?id=2738035