Web Search Engines

Homework 2

Joe Braha and Sonali Seshadri

Introduction

For this assignment, we were asked to compare the effectiveness of the BM25 scoring index and system we previously implemented in assignment two, a dense vector retrieval approach such as a Hierarchical Navigable Small World (HNSW) graph-based vector search index, and a reranking approach that combined the first two systems. We were given a subset of the original MS-MARCO collection of documents, files of queries to evaluate with our systems, files containing vector embeddings of the documents and queries, and query relevance judgment (qrels) files to use for evaluating our own results. Given these files, we were asked to use three different metrics to evaluate our results for the queries using the three different search systems. We chose to use the open source ranx library to implement the TREC standard evaluation, as it seemed to be actively maintained, with over 400 stars on GitHub (more than can be said about alternate options considered).

Search Systems

BM25

We implemented the BM25 scoring system in our previous assignment, in three separate structures: a parser, an index generator, and a query processor. We filtered the collection of documents down to the provided subset using a short Rust program (filter_docs), and fed that smaller collection into our parser, which generated a file of postings. Once the postings were sorted, we used our index generator to create a new compressed index file and lexicon. As a side note, our index generator and processor now compress frequencies instead of truncated impact scores, to correctly implement quantization.

We also implemented batch query processing, so our processor could take in a file of queries and return the results in a file, instead of printing to the terminal for the user. To get the BM25 results for the queries intended for evaluation in this assignment, we wrote a small Python script that would take in a qrels file and a queries file and output a file containing only the queries that had an ID in the qrels file. We ran this program on queries.dev.tsv and qrels.dev.tsv,[1] on queries.eval.tsv and qrels.eval.one.tsv, and on qrels.eval.two.tsv. At the end, we had three files of queries: sorted_queries_dev, sorted_queries_one, and sorted_queries_two.

We ran our batch query processing on these queries, selecting the disjunctive mode–as we thought that was more reasonable for these kinds of queries in order to increase the volume of documents returned and not have documents rejected due to the absence of things like question words and filler words–and requesting the top 100 results. This process was extremely quick, and took less than a minute for the bigger dev queries file (1000 queries), and less than 20 seconds each for the smaller eval one (43 queries) and two (53 queries) query files.

---

[1] We hit an error where qrels.dev didn't have the trec format legacy 0 column, so it was added with `awk '{print $1, "0", $2, $3}' qrels.dev.tsv | columns -t > fixedqrels.dev`.

HNSW

The faiss open source library was used to perform the vector-based searching. The document and query vector embeddings were loaded from the provided h5 files, and the documents are built into a faiss.IndexHNSWFlat with provided parameters for (M, ef_construction, ef_search). Then the queries were queried with index.search() for each of the three files of queries with a few different parameters, as detailed below.

Reranking

The reranking method combined both indexing methods above: the preprocessed BM25 results were loaded, and for each query, the query embedding was loaded from the array of query embeddings, and the embeddings for the documents it returned are collected into a mini HNSW index, and that index is queried.

Results

HNSW Parameters

- M
    - The number of bidirectional links created for each new element during the construction of the graph
    - Greater M values can improve the accuracy of the search, but also increases the amount of memory used and the time it takes to construct the index
    - We tested m = {4, 8)
- k
    - The number of nearest neighbors to retrieve for each query
    - We used k = 100
- ef_search
    - The size of the dynamic list for the nearest neighbor search – this impacts the balance of search speed and accuracy. As you increase ef_search, the accuracy of the search increases but so does the time needed to complete the search
    - This value should generally be at least double the value of k
    - We tested ef_search = {50, 200, 400}
- ef_construction
    - The size of the dynamic list for the construction of the graph – similarly to ef_search, this parameter controls the tradeoff between construction speed and accuracy
    - We tested ef_construction = {50, 200, 400}

Metrics - How to Interpret

- MRR@10 (Mean Reciprocal Rank at 10): Measures the average of the reciprocal ranks of the first relevant result. Higher values indicate better performance.

- Recall@100: Measures the proportion of relevant documents retrieved out of the total relevant documents. Higher values indicate better performance.
- MAP (Mean Average Precision): Measures the mean of the average precision scores for each query. Higher values indicate better performance.
- NDCG@10 and NDCG@100 (Normalized Discounted Cumulative Gain): Measures the ranking quality, taking into account the position of relevant documents. Higher values indicate better performance.

Round 1

M = 4, k = 100, ef_search = 50, ef_construction = 50

Results for Dev

BM25

- mrr@10: 0.36176825396825396
- recall@100: 0.7545
- map: 0.36374600246369476

HNSW

- mrr@10: 0.3169738095238095
- recall@100: 0.5114166666666666
- map: 0.31309217725703764

Rerank

- mrr@10: 0.5405837301587301
- recall@100: 0.752
- map: 0.5338526162002742

Reranking was the best system here. Reranking got the highest MRR, the second highest for Recall, and the highest for MAP. This means the reranking method was better at ranking the first relevant results and had overall better precision. BM25 had the highest Recall, only slightly higher than reranking, which means it retrieved the most relevant documents out of the total relevant documents, and HNSW performed the worst across all metrics.

Results for Eval One

BM25

- ndcg@10: 0.44079363540247263
- ndcg@100: 0.48733381445561447
- mrr@10: 0.8079734219269102

HNSW

- ndcg@10: 0.47836157589623884

- ndcg@100: 0.4417476359587856
- mrr@10: 0.686046511627907

Rerank

- ndcg@10: 0.6825876697856331
- ndcg@100: 0.5506747933559709
- mrr@10: 0.9573643410852715

The rerank method performed the best in all three, indicating it was better at ranking relevant documents at the top positions and overall more precise. BM25 performed much better than HNSW when evaluating with MRR, did slightly better than HNSW when evaluating with NDCG@100, but did slightly worse when evaluating with NDCG@10. This indicates that BM25 was much better at ranking the most relevant document as the first result, but the overall rankings were not ordered as well. As HNSW only outperformed BM25 when evaluated with NDCG@10, we know that HNSW's top ten ranking more closely resembled the ideal order, even if its number one result was not as frequently the most relevant result.

Results for Eval Two

BM25

- ndcg@10: 0.4652332494860899
- ndcg@100: 0.46084284820350174
- mrr@10: 0.7816358024691359

HNSW

- ndcg@10: 0.5260897389563153
- ndcg@100: 0.4749265156747673
- mrr@10: 0.7341269841269841

Rerank

- ndcg@10: 0.6306258978325805
- ndcg@100: 0.5251329114726595
- mrr@10: 0.9203703703703704

With Eval Two, rerank again was the highest performing system, but BM25 and HNSW were more closely tied in performance. Again, HNSW did better than BM25 with the top ten results, but it also did slightly better with the top 100 results, and the discrepancy between HNSW and BM25's MRR values was smaller here, even if BM25 still did better.

Overall Performance

Processing all queries for all vector-based query methods combined took around 6 minutes, and used around 3.6 GB of memory.

The rerank method consistently performed the best across most metrics, especially in terms of MRR@10 and NDCG@10, indicating it was effective at ranking relevant documents at the top positions. The BM25 method performed almost as well in terms of Recall@100 in the dev set and had balanced performance in the eval sets but was generally outperformed by the rerank method. The HNSW method performed the worst in the dev set and had mixed performance in the eval sets, with higher NDCG@10 compared to BM25. As lower values for the parameters of HNSW generally result in lower accuracy and better search time, these results are in line with what we expected. When the accuracy of HNSW is low, reranking BM25 results with HNSW can improve the quality of the BM25 results so they outperform HNSW using the larger index.

Round 2

M = 8, k = 100, ef_search = 400, ef_construction = 400

Results for Dev

BM25

- mrr@10: 0.36176825396825396
- recall@100: 0.7545
- map: 0.36374600246369476

HNSW

- mrr@10: 0.5760039682539682
- recall@100: 0.9165833333333333
- map: 0.5747570761086315

Rerank

- mrr@10: 0.5420837301587301
- recall@100: 0.7535
- map: 0.5354304714967627

HNSW performed the best in all three metrics here, and did significantly better than BM25 in all three, but only significantly outperformed reranking in the Recall@100 metric. When we constructed the index out of all of the documents rather than a subset provided by a BM25 search, a nearest neighbors approach was more effective at retrieving the most relevant documents out of the total relevant documents. We can see here that the recall capabilities of rerank are strictly limited by BM25's recall. Reranking strongly improved the accuracy of the ranking of the first relevant result as well as the overall precision of the query results, but there's no way to improve the number of retrieved relevant documents simply by reranking them.

Results for Eval One

BM25

- ndcg@10: 0.44079363540247263
- ndcg@100: 0.48733381445561447
- mrr@10: 0.8079734219269102

HNSW

- ndcg@10: 0.6983574133004823
- ndcg@100: 0.6531723767653529
- mrr@10: 0.9534883720930233

Rerank

- ndcg@10: 0.6825876697856331
- ndcg@100: 0.5610412799345004
- mrr@10: 0.9573643410852715

Overall, HNSW performed the best here. NDCG measures ranking quality, and the NDCG values for 10 results were only slightly higher for HNSW than for rerank, but both HNSW and rerank's NDCG@10 values were considerably higher than BM25's. The MRR values were generally much higher for all three systems here, but again HNSW and rerank were significantly ahead of BM25. HNSW outperformed BM25 and rerank in NDCG@100, with a much greater discrepancy between HNSW and rerank than for the other two metrics. This shows that HNSW was better at accurately ranking more relevant documents, likely because its index was built on all of the documents. If we consider the ranking quality of the top 10 results more important than the ranking quality of the top 100 results, then HNSW and rerank should be considered more evenly matched. This might be the case if a user is unlikely to look past the first ten provided results.

Results for Eval Two

BM25

- ndcg@10: 0.4652332494860899
- ndcg@100: 0.46084284820350174
- mrr@10: 0.7816358024691359

HNSW

- ndcg@10: 0.6631622546365685
- ndcg@100: 0.6373373394484294
- mrr@10: 0.923015873015873

Rerank

- ndcg@10: 0.6306258978325805
- ndcg@100: 0.5313175808979411
- mrr@10: 0.9203703703703704

Once again, HNSW performed better than BM25 and reranking in all metrics, but had a very small advantage over reranking. Just like the evaluation for eval one, HNSW most outperforms rerank in NDCG@100, indicating that the needs of the user should be considered when comparing HNSW with rerank here.

Overall Performance

Processing all queries for all vector-based query methods combined took around 9 minutes, and used around 3.6 GB of memory.

For eval one, there was less of a discrepancy between BM25 and HNSW/rerank for the MRR metric, but all three systems generally had much higher MRR values than for the dev set. Additionally, the NDCG values for 10 and 100 results were all generally higher than the MAP values for the dev set, so this generally shows that all three of the implemented systems were more effective on the eval one set. The eval one set has considerably fewer queries to search for than the dev set – eval one has 43 queries, and dev has 1000. This could be the cause of the improvement in precision and ranking quality, or it could be specific to the queries in the set.

Round 3

M = 8, k = 100, ef_search = 200, ef_construction = 200

Results for Dev

BM25

- mrr@10: 0.36176825396825396
- recall@100: 0.7545
- map: 0.36374600246369476

HNSW

- mrr@10: 0.573559126984127
- recall@100: 0.9115833333333333
- map: 0.5726314083860036

Rerank

- mrr@10: 0.5420837301587301
- recall@100: 0.7535
- map: 0.5354304714967627

Results for Eval One

BM25

- ndcg@10: 0.44079363540247263
- ndcg@100: 0.48733381445561447
- mrr@10: 0.8079734219269102

HNSW

- ndcg@10: 0.6983574133004823
- ndcg@100: 0.6544663977352406
- mrr@10: 0.9534883720930233

Rerank
- ndcg@10: 0.6825876697856331
- ndcg@100: 0.5610412799345004
- mrr@10: 0.9573643410852715

Results for Eval Two

BM25
- ndcg@10: 0.4652332494860899
- ndcg@100: 0.46084284820350174
- mrr@10: 0.7816358024691359

HNSW
- ndcg@10: 0.6643404173809936
- ndcg@100: 0.6365752384844894
- mrr@10: 0.923015873015873

Rerank
- ndcg@10: 0.6306258978325805
- ndcg@100: 0.5313175808979411
- mrr@10: 0.9203703703703704

Overall Performance

Processing all queries for all vector-based query methods combined took around 8 minutes, and used around 3.6 GB of memory.

The results from Round 3 are almost identical to the results from Round 3. There are 6 exceptions: HNSW MRR for dev, HNSW recall for dev, HNSW map for dev, HNSW NDCG@100 for eval one, HNSW NDCG@10 for eval two, and HNSW NDCG@100 for eval two. The differences are very small, and only show up in the third decimal point onwards. For HNSW in dev, the metrics are slightly better in Round 2 than in Round 2. For HNSW in eval one, the NDCG@100 value is slightly better in Round 3. For HNSW in eval two, NDCG@10 is better in Round 3, and NDCG@100 is better in Round 2. These differences are so slight that we do not think they are significant. It seems like lowering the ef parameters from 400 to 200 did not have a significant impact.

Conclusions

After evaluating our results, we have found that when the parameters for constructing and searching an HNSW index are lower, fetching the results of a BM25 search and then constructing smaller HNSW indexes built on those BM25 results to rerank the results provides us with more precise and overall higher quality rankings than HNSW. BM25 never provides the highest quality results out of the three systems and reranking always improves those results, but the recall values of reranking evaluations is always limited by the recall of the BM25 evaluations. This makes sense, as reranking only reorders the BM25 ranking so that it's closer to an ideal ranking, and it cannot change the amount of relevant documents it receives. Once we increased the construction and search parameters for HNSW, it consistently outperformed BM25 and reranking, but the advantage over reranking was also consistently small. With a larger set of documents to index, HNSW would need to handle a very large number of points and could become very expensive. Since the tradeoffs in performance are relatively minimal for reranking when you compare it to HNSW, it seems likely that using the two-stage reranking approach of BM25 and HNSW would make more sense, as a query against a BM25 index runs quickly and can reduce the overhead of performing a search on a large vector index. The biggest advantage HNSW had over reranking seemed to be in NDCG@100, and in a real world setting, the average user of a search engine is probably much less frequently accessing search results past the top 10 or 20 results, so if an HNSW search became too expensive, we think it would make more sense to use reranking.