CS 469 Information Retrieval Assignment 2

Maheswar Bora
2021A7TS1505H

Aayush Kumar Singh
2021A7PS0430H

Dhananjay Shalley
2021A7PS0401H

Atharva Chikhale
2021A7PS2752H

April 28, 2024

Abstract

This report presents the results of experiments and implementations of vairous information retrieval techniques using the NFCorpus and GENA datasets.

Contents

1 Dataset

1.1 NFCorpus

This is our primary dataset containing documents, queries, query-document relevance values etc.We have noticed that for that the relevance scores are heavily unbalanced and bias towards 2. Hence, many of the evaluation metrics seem skewed. One way to mitigate this issue is to employ a class balancing technique.

1.2 GENA Knowledge Graph

The GENA Knowledge Graph illustrates connections between nutrition and mental well-being. It was derived from Pubmed biomedical abstracts. It consists of entities such as chemicals, foods, and health conditions linked by semantic relations. Example triples include vitamin K's effects on anxiety and the role of B12 in cognitive decline.

2 Problem Statement

The objective of this project is to experiment on various retrieval and ranking models popularly used in information retrieval systems.

3 Experiments

3.1 Experiment 1: Indexing the Datasets

Details on how the NFCorpus and GENA datasets were indexed. We first do preprocessing to tokenize the text, converting it to lowercase, remove stop words, and stemm the remaining words. This ensures different forms of the same word are treated as the same term. We build the inverted index using our custom function. It uses **defaultdict** from python to store the inverted index, where keys are the terms, and the values are lists of tuples (doc id, term frequrency). It also stores the doc length.

3.2 Experiment 2: Vector-based Models

Implementation of vector space models including nnn, ntn, and ntc notations. We have 3 functions to create the document vectors:

- `vectorize_nnn()`: vectors with raw-term frequencies without normalization.
- `vectorize_ntn()`: vectors with term-frequency normalized by max frequency in the document.
- `vectorize_ntc()`: vectors with normalized TF-IDF (Term Frequency-Inverse Document Frequency) and cosine normalization.

Results:
Example nnn vector for one document:
Document ID: MED-1, Vector: 'birth': 7, 'weight': 6, 'head': 4, 'circumfer': 4, 'prenat': 2, 'exposur': 6, 'acrylamid': 11, 'matern': 3, 'diet': 2, 'european': 2, 'prospect': 2,

.....
'among': 1
Example ntn vector for the same document:
Document ID: MED-1, Vector: 'birth': 0.6363636363636364, 'weight': 0.5454545454545454, 'head': 0.36363636363636365, 'circumfer': 0.36363636363636365, 'prenat': 0.18181818181818182, 'exposur': 0.5454545454545454, 'acrylamid': 1.0, 'matern': 0.2727272727272727, 'diet': 0.18181818181818182, 'european': 0.18181818181818182, 'prospect': 0.18181818181818182,
.....
'among': 0.09090909090909091
Example ntc vector for the same document:
Document ID: MED-1, Vector: 'birth': 0.3286342409838593, 'weight': 0.14780981556236208, 'head': 0.2146055559188946, 'circumfer': 0.1818837614171737, 'prenat': 0.10344620859001998, 'exposur': 0.13297958167838947, 'acrylamid': 0.6532796442059445, 'matern': 0.13422440807227717, 'diet': 0.02688365000543877, 'european': 0.0818063504819383, 'prospect': 0.050689844205051744,
.....
'among': 0.018189583317436243

3.3 Experiment 3: Rocchio Feedback Algorithm

Application of the Rocchio feedback algorithm for query expansion using pseudo-relevance feedback. In pseudo relevance feedback we retrieve a set of documents from the user's intitial query. Then we take the top ranked documents (in our case top 10) to refine the query using rocchio algorithm. Finally, we retrieve the documents again using the refined query. Rocchio Algorithm:

$$\begin{aligned}\vec{q_{new}} &= \alpha\vec{q_0} + \beta\frac{1}{k}\sum_{i=1}^{k}\vec{d_i} - \gamma\frac{1}{n-k}\sum_{j=k+1}^{n}\vec{d_j} \\ &= \alpha\vec{q_0} + \beta\vec{d_{rel}} - \gamma\vec{d_{irrel}}\end{aligned}$$

In our case γ is 0 because for pseudo relevance feedback we do not have any irrelevant documents. The top-k documents are takens as the relevant documents.

Results tested on test.queries.ids
Average NDCG@3 before applying Rocchio: 0.2746
Average NDCG@3 after applying Rocchio: 0.2645

Average NDCG@10 before applying Rocchio: 0.1660
Average NDCG@10 after applying Rocchio: 0.1607

3.4 Experiment 4: Probabilistic Retrieval

Here we use the probabilistic models for ranking the document specifically the Language Model and BM-25 models.

3.4.1 Language Model with Jelinek-Mercer Smoothing

We used Language Model with Jelinek-Mercer smoothing to handle the zero probabilitty issue of unseen words. This algorithm estimates the probability of a term in a document,

combining the term frequency in the document with the term's overall frequency in the corpus. The formula for this model is:

$$P(t|D) = (1 - \lambda) \cdot \frac{tf(t, D)}{|D|} + \lambda \cdot \frac{tf(t, C)}{|C|}$$

We took:$\lambda = 0.1$

3.4.2 BM25

BM25 is a bag of words retrieval function ranking documents. It is based on the inverted document frequency (idf) of each query term and the term frequency-normalization in documents. The formula for BM25 is:

$$\text{score}(D, Q) = \sum_{t \in Q} \text{IDF}(t) \cdot \frac{tf(t, D) \cdot (k_1 + 1)}{tf(t, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})}$$

We chose default parameter values of rank_bm25 library: k1=1.5, b=0.75, epsilon=0.25

3.4.3 Evaluation on Probablistic retrieval

We used precision and recall for evaluation:

$$\text{Precision} = \frac{\text{No. of relevant documents retrieved}}{\text{Total no. of docs retrieved}}$$

$$\text{Recall} = \frac{\text{No. of relevant docs retrieved}}{\text{Number of relevant docs}}$$

3.4.4 Comparison between LM and BM25

Query ID	**LM Precision**	**LM Recall**	**LM Top Scores**	**BM25 Precision**	**BM25 Recall**	**BM25 Scores**
PLAIN-10	0.1	0.012658	[-7.59...]	0.1	0.012658	[9.97...]
PLAIN-100	0.0	0.000000	[-11.19...]	0.0	0.000000	[8.06...]
PLAIN-1000	0.0	0.000000	[-6.99...]	0.0	0.000000	[18.03...]
PLAIN-1002	0.6	0.352941	[-5.74...]	0.6	0.352941	[21.12...]
PLAIN-1003	0.0	0.000000	[-9.82...]	0.0	0.000000	[10.37...]
PLAIN-1004	0.3	0.053571	[-6.67...]	0.3	0.053571	[13.28...]
PLAIN-1005	0.3	0.500000	[-8.12...]	0.3	0.500000	[11.29...]
PLAIN-1006	0.5	0.086207	[-2.71...]	0.5	0.086207	[8.27...]
PLAIN-1009	0.5	0.625000	[-3.08...]	0.5	0.625000	[12.29...]

Query ID	LM Precision	LM Recall	LM Top Scores	BM25 Precision	BM25 Recall	BM25 Scores
PLAIN-1010	0.1	0.011628	[-24.95...]	0.1	0.011628	[5.34...]
PLAIN-1011	0.1	0.035714	[-24.95...]	0.1	0.035714	[5.34...]
PLAIN-1012	0.0	0.000000	[-17.00...]	0.0	0.000000	[10.99...]
PLAIN-1013	0.7	0.035533	[-2.68...]	0.7	0.035533	[9.58...]
PLAIN-1014	0.0	0.000000	[-4.22...]	0.0	0.000000	[9.88...]
PLAIN-1015	0.6	0.045113	[-6.06...]	0.6	0.045113	[12.90...]
PLAIN-1016	0.7	0.033019	[-2.53...]	0.5	0.023585	[8.09...]
PLAIN-1019	0.3	0.125000	[-2.60...]	0.3	0.125000	[13.90...]
PLAIN-1020	0.0	0.000000	[-23.02...]	0.0	0.000000	[0.0...]
PLAIN-1021	0.6	0.009160	[-2.48...]	0.6	0.009160	[4.98...]
PLAIN-1022	0.1	0.333333	[-3.91...]	0.1	0.333333	[12.28...]

3.4.5 Observations

LM Top scores refers to logarithimic scores calculated for the top documents in relevance to the query, indicating probability of query terms given a document summed across Both models have similar precision and recall.

3.5 Experiment 5: Entity-based Retrieval Models

Integration of the GENA knowledge graph in retrieval models to enhance entity-based search.

- Spacy's NLP model is used to extract the entities from both queries and documents. The queries are converted to lower case to ensure normalization.
- Here we implement a bag of enities based retrieval model.
- We transform the entities into tf-idf based model
- We compute the similarity between the query and document vector's entities.
- We rank the docs based on the similarity score

3.5.1 Results

Top-5 similarity scores for query PLAIN-2430: "preventing brain loss with b vitamins ?"
Document ID: MED-4872, Score: 0.30151134457776363
Document ID: MED-980, Score: 0.2886751345948129

Document ID: MED-4721, Score: 0.2886751345948129
Document ID: MED-1989, Score: 0.25
Document ID: MED-1999, Score: 0.25

3.6 Experiment 6: Query Expansion using Knowledge Graphs

Development of a query expansion technique that leverages relations in the knowledge graph.

- Spacy's NLP model is used to extract the entities from both queries and documents. The queries are converted to lower case to ensure normalization.
- For query expansion we first extract entities from the original query
- We then use the entities to find the related terms in the knowledge graph.
- Initially we thought of using the synonyms and entity relation but since the GENA knowledge graph did not contain many entities that were extracted from spacy we decided to use all the relations.
- Since some of the objects in the knowledge graph are sentences we decided to use entity extraction again ('re-extraction') to get the final expanded query entities.

3.6.1 Results

Original Query Entities: ['preventing', 'brain loss', 'vitamins']

Expanded Query Entities: ['preventing', 'cycle mechanism depression', 'glucose homeostasis hyperkinetic disorder', 'dysfunction', 'metabolic disorders', 'deficit', 'minerals', 'vitamins', 'hyperkinetic disorder', 'insulin resistance', 'lipoprotein resistance', 'folate metabolism', 'group', 'patients', 'chronic alcoholism', 'assessed', 'nutritional status', 'minerals', 'vitamins', 'compared', 'control subjects', 'psychiatric disorders', 'deficiency', 'mineral nutrients', 'vitamins', 'vitamins', 'neurogenesis', 'defense mechanism', 'working', 'neurons', 'metabolic reactions', 'neuronal survival', 'neuronal transmission', 'defense', 'mechanism', 'obesity', 'putative biomarkers', 'regulation', 'metabolic processes', 'vad', 'pathway analysis', 'biosynthesis', 'amino acids', 'metabolic pathways', 'correlated', 'occurrence', 'vad', 'vitamins', 'amino acids', 'derivatives', 'vad diagnosis', 'treatment monitoring', 'significance', 'tcm', 'classification', 'hengqing', 'prescription', 'treatment', 'vad',
.....
'cognitive impairment', 'associated with', 'consumption', 'food groups', 'fresh vegetables', 'fruits', 'poultry', 'eggs', 'plant oil', 'nutrients', 'protein', 'fat', 'fiber', 'antioxidant vitamins', 'biochemical', 'mthfr', 'vitamins', 'cofactors', 'remethylation', 'homocysteine', 'folate', 'homocysteine metabolism', 'vitamins', 'impaired', 'course', 'psychotic symptoms', 'persecutory delusions']

Top-5 similarity scores for query PLAIN-2430: "preventing brain loss with b vitamins ?"
Document ID: MED-4470, Score: 0.2694935014798511
Document ID: MED-4012, Score: 0.26949350147985107
Document ID: MED-4029, Score: 0.26949350147985107

Document ID: MED-1059, Score: 0.22997745696362096
Document ID: MED-1455, Score: 0.22997745696362096

After re-extraction:
Document ID: MED-4872, Score: 0.41041992544780836
Document ID: MED-980, Score: 0.29043920683391916
Document ID: MED-951, Score: 0.2708377590770591
Document ID: MED-4928, Score: 0.26632348027748143
Document ID: MED-4470, Score: 0.2526566376179321

3.7 Experiment 7: Learning to Rank Models

Here we implement pointwise, pairwise, and listwise learning to rank models.

3.7.1 Pointwise

Here the learning to rank model treats the task of ranking as a regression or clasification problem taking individual documents as input. Each query-document pair is considered independent of others.

- The cosine similarity is calculated between the tf-idf vectors of the query and the document, serving as features for the model.
- Here we used scikit-learn's RandomForestClassifier as our model. Here we treated predicting relevance as classification problem.
- Further improvements can be made by using query expansion techniques, or deep learning based embeddings using models like BERT.

Result:
NDCG@10: 0.6732186732186731

3.7.2 Pairwise

This approach takes pairs of documents as input to deternming which one is more relevant to the query.

- Here we generate the document pairs from the same query to compare two documents at a time.
- In our prepared dataset if the first document has higher relevance than the second then the label is 1, and 0 otherwise.
- We use SVM, or SVC with linear kernel, to classify whether the first document in a pair is more relevant than the second. Further improvements can be made such as using a more complex model like lambda rank.

Result:
We could not take all possible pairs due to time and memory constraints. So we limited it to 10 pairs per query. Hence we are unable to achieve good results. However, with increase in pairs we could see increase in NDCG scores.
NDCG@10 1 pair per query: 0.07336392209936006
NDCG@10 10 pairs per query: 0.35895421017163476

3.7.3 Listwise

Here we used the ListNet Approach to train the model.

- We create a list of tuples containing the query text, the associated document text and their relevance scores.
- We use SkLearn's tf-idf vectorizer to get the features.
- We implemented the model, inspired by ListNet, from scratch using PyTorch. The model consists of Linear layers with ReLU activaton function. This model tries to score the documents for a given query, thereby optimizing the order of the documents.
- We employed cross entropy loss between the predicted scores and the true relevance scores transformed into probability distributions. Here we directly address the entire list of documents related to a query.
- Loss formula used:

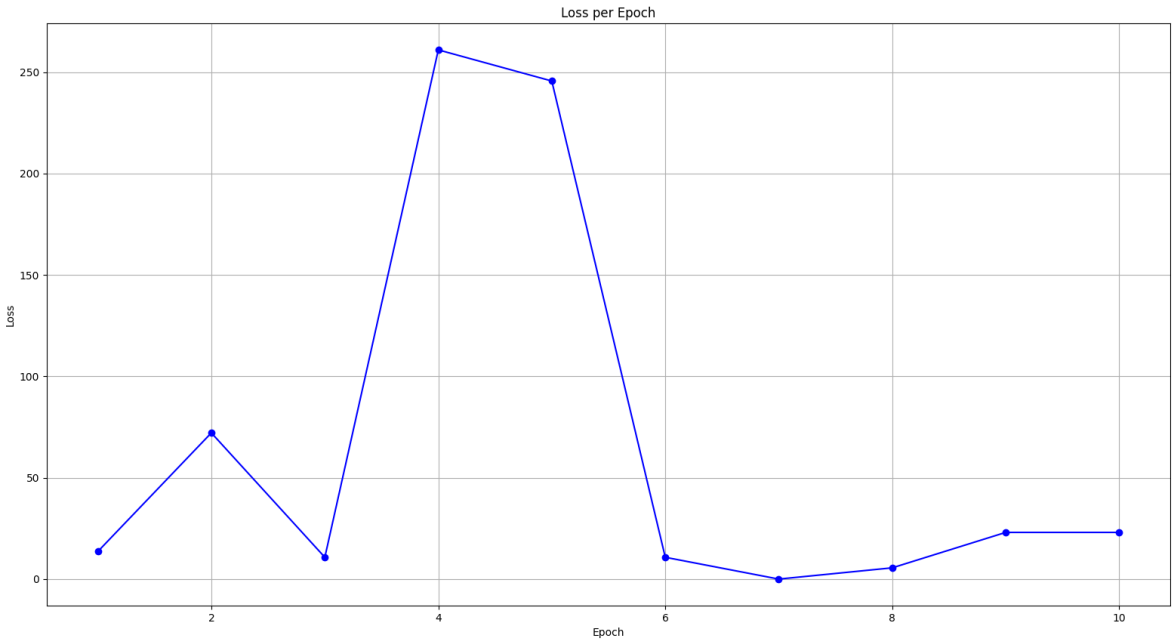
$$\text{Loss} = -\sum_{i} P(y_i) \log P(z_i)$$

  where $P(y_i)$ and $P(z_i)$ are the probability distributions of the true and predicted scores, respectively.

Result:
NDCG@10: 0.875

3.7.4 Observations

We conclude that Listwise approach is the best approach with the highest NDCG score. This is followed by pointwise approach which gives us a simple yet effective and efficient algorithm giving us a relatively moderate NDCG score. The pairwise approach is computationally expensive hence we conclude this approach is not optmimal for this dataset.



3.8 Experiment 8: Enhancements to improve NDCG scores

We have tried vairous optimizations to improve the NDCG scores in most of our experiments.

- In the data preprocessing side we use stemming, lemmatization, lower casing etc using nltk.
- For the knowledge graph experiments we use a vocabulary of spaCy, `en_core_sci_md-0.5.4`, that is specific to bio and medical related data to improve the entity extraction.
- For the learning to rank model we changed the vectorizer. Instead of using sklearn's tf-idf vectorizer we experimented with a BERT based vectorizer. This will transform the features. However this increases the memory consumption signifcantly so, we were not able to train for multiple epochs on time.

4 Conclusion

In this assignment we successfully learnt to implement various retrieval and ranking strategies in information retrieval.

5 References

References

[1] BITS-Hyderabad Information Retrieval Slides.

[2] PyTorch Documentation

[3] Scikit-Learn Documentation

[4] Tensorflow Learning to Rank.

[5] Towards Datascience: *What is Learning to Rank? A Beginner's Guide to Learning to Rank Methods.*
`https://towardsdatascience.com/what-is-learning-to-rank-a-beginners-guide-to-lear`

[6] Microsoft Research: *Learning to Rank: From Pairwise Approach to Listwise Approach*
`https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2007-40.pdf`