IR ASSIGNMENT 2

GROUP 2

Pranathi Voora, Srivant Vishnuvajjala, Vivan Jain, Samarth Pawan

3.1: Indexing the Data Sets

Indexing of the datasets was recommended to be done using Lucene as an indexer. However, we have adopted our own indexing techniques for each experiment and customised it as per the need of the task. Few of the common indexing techniques used throughout all experiments are as follows: - Stop word removal, Stemming and Lemmatising - Using Inverted Indexes to calculate term frequency and document frequencies - Constructed a matrix of the following formats - entity: list of docs and query: list of entities [For Exp 5, 6, only] - Used String Lower-Casing of queries and docs prior to search retrieval - Tokenization: Breaking down text into individual tokens (words) using the NLTK word-tokenize function. - Text Normalization: Normalizing the text by removing unwanted characters and punctuation while keeping hyphens.

25963 words in df.txt after indexing.

3.2: Vector Based Models

The terms nnn, ntn and ntc in the context of vector-based retrieval models refer to different combinations of normalization and weighting techniques applied in Information Retrieval. The task asked us to implement all 3 of them as follows:

NNN

This term stands for Natural, Natural, No normalization. The term frequency (TF) is the count of how many times a term appears in a document. The document frequency (DF) is 1 for each term, indicating no inverse document frequency (IDF) adjustment. This approach doesn't normalize the resulting scores, hence there are no adjustments for document length or term dispersion. This is the simplest form of a vector-based retrieval model, of the three.

NTN

This term stands for Natural, Term frequency, No normalization. In this configuration, document frequencies are adjusted with a logarithmic scale to compute the inverse document frequency (IDF). The normalization remains none. This approach adjusts the scores based on term dispersion across the corpus but doesn't normalize the vector magnitude.

NTC

This term stands for Natural, Term frequency, Cosine normalization. Similar to NTN, it calculates the term frequency, uses the logarithmic scale for inverse document frequency. However, it incorporates cosine normalization, where each document vector is adjusted to have a magnitude of 1 (unit vector). This normalization technique considers the length of documents and makes the scores more comparable across documents of varying lengths.

Table 1: NDCG

Query File	NNN	NTN	NTC
train-titles	0.2457	0.2801	0.3743
train-nontopic	0.2847	0.3493	0.3452
train-viddesc	0.2897	0.3483	0.3127
train-vidtitles	0.2879	0.3557	0.3539
test- titles	0.2528	0.2864	0.3756
test - nontopic	0.2851	0.3475	0.3484
test - viddesc	0.2862	0.3421	0.3117
test - vidtitles	0.2797	0.3437	0.3534
dev - titels	0.2395	0.2676	0.3650
dev - nontopic	0.2846	0.3358	0.3445
dev - viddesc	0.2742	0.3311	0.3072
dev - vidtitles	0.2802	0.3324	0.3469

Our Approach

To perform computations using vector based models, we have first created a tf-index which basically is a parsed version of the indexing we did in the first experiment of the model. After that we have parsed the queries and separated every term in the query and stored in a 2d array. Based on the indexes, we have written 3 functions: score-documents-nnn: Implements the 'nnn' strategy, where raw term frequencies (tf) from the tf-index dictionary are used to score documents based on how often query terms appear in each document. No normalization or inverse document frequency (IDF) adjustments are applied. score-documents-ntn: Utilizes the 'ntn' strategy. It employs raw term frequencies and adjusts the scores by applying IDF, calculated using the logarithmic scale based on document frequencies (df) obtained from the tf-index. No vector normalization is applied. score-documents-ntc: This function scores documents using the 'ntc' strategy, which involves both the term frequency and IDF adjustments from 'ntn' and applies cosine normalization to the score vectors, turning them into unit vectors. This normalizes for document length and is intended to make the scores more comparable across documents. Lastly, the function write-scores-to-file then writes these scores to a file, formatting them for evaluation, presumably with metrics like nDCG.

How do they compare?

Ideally, NTC is supposed to perform better than the other two methods. Without normalization, longer documents might appear more similar to each other due to their length, which may not reflect their semantic similarity. Cosine normalization removes this bias by scaling the vectors to have a unit length. Our results predominantly observe the same as well. However, for few queries such a video description, our NTN score dominates. The following is our explanaation for the anomalous case of NTN dominating over NTC.

Our Results

- In the NNN configuration, where no normalization or IDF adjustments are made, the method relies purely on the raw frequency of terms in the documents. Given the medical dataset's specificity, if many documents have a similar length and use the technical terms consistently, NNN might inadvertently boost documents that merely mention the query terms frequently, regardless of their true relevance. The slightly better performance of NTC over NNN in some cases could be because while NTC does normalize for document length (potentially reducing the weight of longer documents that don't match as well), it doesn't adjust for term importance as effectively as NTN.

- The corpus might contain a mix of general medical terms and highly specific terms. NNN does not distinguish between common and rare terms, possibly leading to a scenario where documents that happen to have a higher count of common terms score well, even if they are not as relevant. NTC attempts to address this with cosine normalization, but without the IDF weighting provided by NTN, it is possible that it may not sufficiently down-weight the impact of common terms or up-weight important specific terms. We believe this is why NTN is outperforming both NNN and NTC in some cases.

- Considering the sample queries, which are quite precise and likely to correspond with exact matches in the medical documents, NTN outperforms due to the IDF factor amplifying the significance of rarer, more informative terms. NNN, without this, is more likely to miss subtleties of term significance, while NTC may normalize away the importance of exact term matches due to its focus on document vector normalization.

3.3: Rocchio Feedback Algorithm

The task requires us to implement the Rocchio Feedback Algorithm for Query Expansion using Pseudo-Relevance Feedback.

We use a top k value of 20. Pseudo relevance feedback assumes the top k queries as the most relevant documents. This automates the manual part of relevance feedback.

We have created vectors for the queries and documents using the vocabulary in the dictionary. We applied the Rocchio algorithm formula using relevant documents as the top k from the results from vector space model (NNN).

We then get the modified queries according to the pseudo-relevance feedback for query expansion. Then, we ran NNN again with the new modified queries and received a new ranking of vectors.

Table 2: NDCG for Test Queries

Query File	Rocchio Pseudo Relevance Feedback	Improvement
test- titles	0.2893	0.0365
test - nontopic	0.2951	0.01
test - viddesc	0.2916	0.0054
test - vidtitles	0.2747	0.0115

Observation: As we increased the top k values, the relevance decreased.

Alpha, beta, gamma values

0.93 as alpha value: has a strong emphasis on the original query. It hints that we do not deviate far from the original query.

0.75 as beta value: adjusts towards centroid of relevant documents. It maintains balance between relevance of relevant docs and the query.

0.25 as gamma value: adjust towards centroid of non-relevant documents.

3.4 Probabilistic Retrieval

The task asked us to implement language models with probabilistic retrieval exploring BM-25 and LMs.

BM-25

The task asked us to implement language models with probabilistic retrieval. We used the Okapi BM25 Model to do so. We are using the RSVd formula and using term frequency and document length normalisation. The tuning parameters are set to 1.2 to control document term frequency and 0.75 to control the scaling.

The RSVd, retrieval status value, is calculated by summing the individual contributions of each query term's TF and IDF which are then normalised by the constants mentioned above.

Table 3: NDCG values and BM25 scores

Query Files	BM25 scores
train-titles	0.4366
train-non-topic	0.4454
train-vid-desc	0.4667
train-vid-titles	0.4511
test-titles	0.4320
test-non-topic	0.4441
test-vid-desc	0.4542
test-vid-titles	0.4310
dev-titles	0.4185
dev-nontopic	0.4316
dev-vid-desc	0.4483
dev-vid-titles	0.4207

Language Models

The task asked us to implement language models with probabilistic retrieval. We used the mixture model to implement this. For all the terms in the query, we found the probability that the terms would occur in document. The product of the probability for each of these terms in q query for a document gives us the probability for that query-document pair. This model combines the context of the document with the general usage of words in the entire document collection.

If lambda is higher, terms frequent in documents will be given more weight and if lambda is lower, a collection frequency is given more weight to smooth the effect of specific terms.

NDCG = 0.4566 on test.queries.titles

3.5: Entity-based retrieval models

The task asked us to use any of the entity based retrieval techniques. We have implemented the coordinate matching technique as well as the entity frequency technique. However, both of the ndcg scores resulted to be in the same range. Since entity frequency slightly under-performed by 0.003, we have stuck to coordinate matching to represent our final ndcg scores for all.
Explaining the method used in detail as follows:

Initially we attempted to hold a entity vs documents matrix, with 1 representing that the entity exists in that particular document(column). However, due to high time consumption, we adopted the alternative method, where we stored for each entities, the docID's of documents which it exists in (similar to the concept of adjacency lists in graphs as an alternative to adjacency matrices). This was then written onto a file for query usage. This was optimised to the fullest by integrating the Aho-Corasick algorithm to store the entitities in the form of a Trie for efficient retrieval.
Secondly, we have maintained query: list(entities which query contains), and mapped them to corresponding docs these entities exist in using the entity: list(docID's) from the file. Looping through this we have appended a score of 1 to every query:docID pair. The final scores have been written onto a file in necessary format and have calculated NDCG scores using merged.qrel as a relevance measure.

Table 4: NDCG values and Entity Based Retrieval Techniques (Coordinate Matching)

Query Files	Entity Based Retrieval Techniques (Coordinate Matching)
train-title	0.1919
train-nontopic	0.1930
train-nontopic (entity)	0.1908
train-viddesc	0.2078
train-vidtitles	0.2062
test-title	0.2010
test-title (entity)	0.1990
test-nontopic	0.1977
test-viddesc	0.2132
test-vidtitles	0.2062
test-vidtitles (entity)	0.2001
dev-title	0.1830
dev-nontopic	0.1845
dev-viddesc	0.1925
dev-vidtitles	0.1846
dev-vidtitles (entity)	0.1829

3.6: Query Expansion using Knowledge Graphs

Similar to the previous experiment, we are extracting entities and documents to construct a matrix reflecting entities' presence across the document collection. Initially, documents were processed to concatenate their URL, title, and abstract into cohesive text, upon which we performed standardization and punctuation removal. Using the Aho-Corasick automaton, entities were efficiently extracted from the processed documents.

The aim of this section of the assignment was to improvise ranking score of entity based retrieval using query expansion techniques by making use of relations in the triples representation of the knowledge graph. Upon analysing all the relations, there were 30000+ unique relations, and out of which we deemed only the following as useful.

Relations Data Description

Sentence: This relation contains descriptive sentences about the subject, providing crucial contextual information.
Weight: 1.0
Full-E1/Full-E2: These relations likely represent the full name or description of the subject, which can be important for understanding its identity.
Weight: 0.9
MeSH-E1/MeSH-E2: Medical Subject Headings (MeSH) terms provide standardized vocabulary for indexing and searching biomedical literature. They are highly relevant in medical and healthcare contexts.
Weight: 0.8
Synonyms-1/Synonyms-2: Synonyms are alternative names or terms associated with the subject, which can help capture variations in terminology.
Weight: 0.7
Type-E1/Type-E2: These relations indicate the type or category of the subject, which can provide additional context but may be less informative compared to other relations.
Weight: 0.6
ID-1/ID-2: These relations likely represent identifiers or codes associated with the subject, which may be useful for referencing but might not provide as much semantic information.
Weight: 0.9

Having assigned weights to the objects connected by these relations, we attempted to consider these objects as entities itself and apply the same coordinate matching techniques. However, instead of adding a score of 1 to the query-doc pair, we would be adding a score of the weight assigned.

Incorporating the 'Sentence' Relation

Over and above this, we have also integrated the usage of the relation type "Sentence", which we treated as a the description document for that entity. Instead of assigning a simple weight to this, we attempted to calculate Jaccard similarity of the object and the document, and attempted to add that to the score. Seeing negative results, we attempted to use a TensorFlow library for similarity calculation, which reflected in positive improvement of result.

```
import tensorflowhub as hub
embed = hub.load("https://tfhub.dev/google/universal-sentence-encoder/4")
```

Table 5: NDCG values for Knowledge Graphs with Query Expansion - Incorporated Relations

Query Files	Knowledge Graphs using Query Expansion - Incorporated Relations
train-title	0.1964
train-nontopic	0.1979
train-viddesc	0.2381
train-vidtitles	0.2081
test-title	0.2001
test-nontopic	0.1998
test-viddesc	0.2314
test-vidtitles	0.1924
dev-title	0.1903
dev-nontopic	0.1920
dev-viddesc	0.2381
dev-vidtitles	0.2081

3.6: Learning to Rank models

Merged all queries together with a new merged.qrel.

Pointwise

Pointwise LTR is a method used in information retrieval systems to predict the relevance of documents to a specific query. Pointwise LTR approaches treat the ranking problem as a regression or classification task where each query-document pair is scored independently.

The query ids, the doc ids and the relevance scores are extracted from merged.qrel. Using TensorFlow's StringLookup, these IDs are converted into numerical indices. A relevance score matrix is then initialized. Then a neural network model is constructed with separate input paths for queries and documents. An embedding layer is constructed for the query ids and the doc ids. The predicted relevance is then calculated by finding the dot product between the embedding layers. The model is trained using mean squared error as the loss function, and it learns to predict relevance scores for all query-document pairs. The accuracy of the predictions is evaluated using the Normalized Discounted Cumulative Gain (NDCG) metric.

NDCG	0.1509

Why a lower NDCG than other unsupervised models seen?:
This could be due to the lower model complexity used in our LTR model compared to the supervised models used.

Pairwise

Pairwise learning-to-rank is an approach used in machine learning for information retrieval and recommendation systems that models the preference relations between pairs of items within the same query. Initially, query IDs, document IDs, and relevance scores are extracted from a dataset, converted into arrays, and then used to construct a TensorFlow dataset. In this scenario, query IDs serve as "user IDs" and document IDs as "movie titles".
The dataset is shuffled and divided into training and test sets. In pairwise training, the data is processed to form pairs of documents for each query, where each pair consists of two documents with different relevance scores. A custom RankingModel class is developed, including embedding layers for users and documents, followed by a scoring mechanism using dense layers that predicts which document in each pair is more relevant.The model uses the Pairwise hinge loss model.The predictions are evaluated using NDCG scores.

NDCG	0.0.9723

Listwise

Listwise learning-to-rank is an approach used in machine learning for information retrieval and recommendation systems that models the entire list of items as a single unit for training. First the query IDs, document IDs, and relevance scores are extracted, converting these into arrays and constructing a TensorFlow dataset. This dataset, treating query IDs as "user IDs" and document IDs as "movie titles," simulates a movie recommendation scenario. The dataset is shuffled and divided into training and test sets, which are then processed using listwise sampling. This sampling strategy creates multiple lists of documents per query, each containing a few documents . A custom RankingModel class includes user and document embedding layers and a scoring mechanism using dense layers to predict document relevance. The model uses the ListMLE loss function. The predictions are evaluated using NDCG scores.

NDCG	0.9751

3.6: Out of the Box

Rocchio Improvement

Our improvement involves an expansion on Rocchio Feedback Algorithm where we split the beta and gamma values into levels based on relevance instead of taking just top k. The first k will have a higher beta value as it will hold a larger contribution to the relevant documents. The next k will have a lower beta value to hold some contribution to the relevant documents but not as much as the first k. This continued and the same is done for the bottom k with gamma values.

NDCG score with top k: 0.2893
NDCG score with levels of top k: 0.3051

Recomender Systems for Pointwise Ranking

In a case where we want to expand on the relevance scores for documents for queries where the queries do not have a ranking for the documents, we are using recommender systems. We are using neural networks for our regression model and an mae loss function to compute pointwise ranking of the documents. The model will suggest new documents with its corresponding relevance for queries that have not ranked them before.

This shows a significant improvement in the ndcg score to 0.95 as queries being used have a corresponding ranking associated with them.

System Details

System Memory Used: 16GB RAM
Processor Details: M1 PRO CHIP/ M1 AIR CHIP