CA6005 2022-2023 Assignment 1
Search Engine

[Code repository: https://github.com/thomas236/CA6005-2022-2023-Assignment-1-Search-Engine]

Thomas Sebastian†
MSc Computer Science (Artificial Intelligence)
Dublin City University/University of Galway (Student ID: 21250103)
Ireland
thomas.sebastian3@mail.dcu.ie

ABSTRACT

This report presents the implementation and evaluation of a search engine for the Cranfield dataset as part of the Dublin City University MSc Computer Science (Artificial Intelligence) program, CA6005 Mechanics of Search Assignment 1. The search engine incorporates three different similarity measures: Cosine Similarity, BM25, and Gensim [4]. The code pre-processes the Cranfield dataset by tokenizing, removing stop words, and stemming text. The search engine then computes the TF-IDF matrix for queries and documents for Cosine Similarity, uses the BM25Okapi implementation for BM25, and Gensim for the third method. The engine retrieves the top 100 similar documents for each query and saves the results in separate text files for further evaluation. This report provides an overview of the evaluation of three text retrieval methods: Cosine Similarity, BM25, and Gensim. The evaluation is conducted using the Dyneval tool, which calculates performance metrics such as Mean Average Precision (MAP), Precision@5 (P@5), and Normalized Discounted Cumulative Gain (NDCG) [5] for each method.

INTRODUCTION

The goal of this project was to implement an information retrieval system for the Cranfield collection dataset. The system is based on the Vector Space Model and implements three different methods for indexing and ranking: cosine similarity, BM25, and Gensim. The Vector Space Model is a widely used technique for information retrieval, where the documents and queries are represented as vectors in a high-dimensional space, and the similarity between them is measured by the cosine of the angle between the vectors.

The Cranfield dataset is a collection of documents used for information retrieval research. This code performs several pre-processing steps on the dataset to prepare it for further analysis. The first part of the code deletes any existing folders or files and creates new directories to store the pre-processed data. The second part extracts the queries from the XML file and saves them to individual files in a newly created directory. The third part reads in the raw documents from the Cranfield dataset and extracts the document number and text elements, saving them to individual files in a newly created directory. The fourth part pre-processes the query files by removing punctuation and stop words and stemming the remaining words. Finally, the last part initialises a Porter Stemmer Object to create a Tree/Stem Model.

The code processes both the documents and the queries in the dataset to convert them to a pre-processed format for later use. The pre-processing involves removing stop words, stemming, and converting to lowercase. The pre-processed files are saved in new directories named 'preprocessed_cranfieldDocs' and 'preprocessed_cranfieldQueries'. The code also implements cosine similarity on the pre-processed documents and queries for later use in ranking the relevance of documents to queries.

The first algorithm uses the cosine similarity between queries and documents to find the relevant documents. The pre-processed queries and documents are converted to TF-IDF vectors, and the cosine similarity is calculated between them. The resulting similarities are ranked, and the top 100 ranked documents are written to a file.

The second algorithm is BM25, which is a bag-of-words retrieval model based on the frequency of the query terms in the documents. BM25 considers the term frequency, the document length, and the inverse document frequency of the query terms. Like the first algorithm, the pre-processed queries and documents are used to calculate the scores, and the top 100 ranked documents are written to a file.

The third algorithm uses Latent Semantic Indexing (LSI) to find the relevant documents. LSI is a technique that transforms the original data into a lower-dimensional space and then uses the cosine similarity in the transformed space to calculate document relevance. In this implementation, the pre-processed queries and documents are used to create a dictionary, which is then used to create a corpus. The corpus is then transformed using LSI, and the cosine similarity between the queries and documents is calculated. The top 100 ranked documents are written to a file.

INDEXING

Indexing is the process of creating an index for a collection of documents to enable fast retrieval of relevant documents for user queries. The indexing process includes document analysis, pre-processing, indexing construction, and data structures. In this section, we describe the indexing process for the Cranfield collection.

Document analysis: The Cranfield collection consists of documents in SGML format. The documents contain title, author, bibliographic references, and abstract. The first step in the indexing process is to extract the relevant information from the documents. We used Beautiful Soup [1], a Python library, to extract the “docno” and “text” elements from each document and saved them to a new file.

Pre-processing: The next step is to pre-process the documents to remove noise and reduce the dimensionality of the data. We performed the following pre-processing steps:

a) Case standardization: We converted all the text to lowercase to reduce the dimensionality of the data.

b) Tokenization: We used the word_tokenize function from the nltk library [3] to tokenize the text into words.

c) Stopword removal: We removed stop words from the text to reduce noise.

d) Stemming and lemmatization: We used the PorterStemmer from the nltk library to stem the words and WordNetLemmatizer to lemmatize the words.

The cosine similarity method uses the pre-processed documents and queries to calculate the cosine similarity between them. This method assumes that the term frequency and inverse document frequency of the terms in the document and query are independent and multiplies them together to obtain the final score.

The BM25 method is a more advanced technique that considers the document length, term frequency, and inverse document frequency. This method is known to perform well on long documents and queries.

The Gensim method uses Latent Semantic Indexing (LSI) to convert the pre-processed documents and queries into a low-dimensional space, where the similarity between them is measured using the cosine similarity metric.

RANKING

The ranking component of the system involves the retrieval and ranking of documents based on their similarity to the query. The cosine similarity, BM25[8], and Gensim[11] methods differ in their approach to ranking.

The cosine similarity method ranks the documents based on their cosine similarity scores with the query. The documents with the highest scores are ranked higher. The system first pre-processes the raw documents and queries by tokenizing them, removing stop words and stemming them to create a list of clean and stemmed tokens[10].

The system then uses the TfidfVectorizer from Scikit-learn [12] to create a document-term frequency-inverse document frequency (TF-IDF) matrix for both the queries and documents. The cosine similarity is then computed between each query and each document to create a similarity matrix [7].

The results are saved to a text file where each line contains the query ID, the document ID, the rank, the similarity score, and the run ID. The text file is then sorted in ascending order based on the query ID and the rank. The top-ranked documents are the ones with the highest similarity score.

The system uses a ranking of 100 and a run ID of 140. It also extracts relevant information such as the docno, title, author, bibliography, and text from the raw file to help improve the ranking.

The BM25 method ranks the documents based on their BM25 scores [8]. The documents with the highest scores are ranked higher.

The Gensim method ranks the documents based on their LSI scores [11]. The documents with the highest scores are ranked higher.

The top 100 ranked results for each query were returned for each of 225 queries.

Like cosine similarity, the code first reads pre-processed queries and documents from the provided directories. It uses BM25Okapi and Gensim similarity search to create index objects for the documents. It then computes similarities between queries and documents using the BM25Okapi and Gensim similarity search objects.

The similarities are then sorted in descending order, and the top 100 ranks for each query are saved to a file in the format "query_id iteration doc_id rank score run_id". The rank and score are assigned based on the order of similarity scores, with the highest score receiving a rank of 1 and lower ranks assigned based on the order of similarity scores.

The results file is then read, and the lines are sorted in ascending order of the first column (query_id) and saved to the same file. This is done to make the results file easily readable and interpretable.

The BM25Okapi algorithm is a variant of the BM25 algorithm and is used for ranking documents based on their relevance to a given query. It considers the frequency of a query term in a document, the length of the document, and the length of the corpus. The algorithm also includes a parameter k1 and b, which control how the length of the document and the length of the corpus affect the score.

The Gensim similarity search algorithm is used to find documents like a given query based on the cosine similarity between the document and query vector. The algorithm creates a vector space model of the documents and queries, where each document and query are represented as a vector of term frequencies. The algorithm then computes the cosine similarity between the query and each document vector and returns the top matching documents.

In the BM25 implementation, the pre-processed queries and documents are first collected from the directories. A BM25 object is then created using the pre-processed documents and default parameters of k1=1.2 and b=0.75. Similarities are computed between each query and all the documents using the BM25 algorithm. The similarities are then sorted in descending order, and only the top 100 documents with the highest similarity scores are retained. The results are saved to a file, which is then sorted in ascending order based on the query number.

In the Gensim implementation, a dictionary is first built using the pre-processed documents, and the documents are then converted into bag-of-words format and indexed. Similarities are computed between each query and all the documents using the MatrixSimilarity method in Gensim. The similarities are then sorted in descending order, and only the top 100 documents with the highest similarity scores are retained. The results are saved to a file, which is then sorted in ascending order based on the query number.

In both implementations, the final output files are sorted in ascending order based on the query number.

The choice of BM25 and Gensim algorithms for ranking is motivated by their effectiveness in information retrieval tasks. BM25 is a widely used and effective algorithm for ranking documents based on query terms. Gensim, on the other hand, provides an easy-to-use implementation of the Latent Semantic Indexing (LSI) algorithm, which has been shown to be effective in capturing the semantic meaning of documents.

The choice of data structures for indexing and retrieval of documents is also important for efficient search and ranking. In both implementations, the preprocesseddocuments are converted into a bag-of-words format, which allows for efficient indexing and retrieval of documents. The use of a dictionary in the Gensim implementation allows for efficient mapping of terms to their corresponding IDs, which improves the efficiency of the retrieval process.

EVALUATION

The steps that were taken for generating results from the three evaluation methods are as below:

Any existing folders and files that may be present were deleted and created new folders to store the output files.

1. Processing of query files:

The Cranfield query XML file was read, and the query number and title were extracted. The query title was extracted to a file with the same name as the query number in the cranfieldQueries folder.

2. Processing of document files:

The Cranfield document XML file was read, and the document number were extracted with text elements from each document. The text of each document was saved to a new file with the same name as the document number in the cranfieldDocs folder.

3. Pre-processing of Cranfield documents and queries:

The contents of each query and document file were scanned to remove stop words, punctuation, and convert text to lowercase. It uses tokenization, lemmatization, and stemming to normalize the text. The preprocesseddocuments are stored in the preprocessed_cranfieldDocs folder, and the preprocessedqueries are stored in the preprocessed_cranfieldQueries folder.

Overall, the code prepares the Cranfield collection for use in information retrieval tasks by creating the necessary folder structure, extracting the necessary information from the XML files, and pre-processing the text of the documents and queries.

The Cranfield collection dataset was used to evaluate the performance of the system. The dataset consists of 1400 documents and 225 queries.

The evaluation was done using dyn_eval which is a variation of trec_eval. The benefit of using dyn_eval was that it was distributed as and executable file compared to trec_eval that had to be compiled for it to be usable for this purporse. One of the drawbacks of using dyn_eval was the at the reference evaluation file (qrel) was missing a column ‘iter’. It was necessary to add in this information in the second column to successfully using dyn_eval with parameters qrel and the output results files of the three evaluation models. The evaluation results for the three different information retrieval models, namely Cosine similarity, BM25, and Gensim. Before we discuss the metrics for this dataset it is important to give an overview of the metrics used.

MAP (Mean Average Precision), P@5 (Precision at 5), and NDCG (Normalized Discounted Cumulative Gain) are popular evaluation metrics used in information retrieval to assess the effectiveness of search systems or algorithms.

1. MAP (Mean Average Precision): MAP is a measure that combines both precision and recall, providing a single

number to summarize the quality of a search system. It is calculated by first computing the average precision for each query and then taking the mean of these values across all queries. Average Precision (AP) is the average of the precision values obtained for each relevant document in the ranked results [15] up to the position of that document. MAP is particularly useful for comparing the performance of different search systems or algorithms, as it takes into account both the number of relevant documents retrieved and their ranks.

2. P@5 (Precision at 5): Precision at 5 (P@5) is a metric that measures the proportion of relevant documents among the top 5 retrieved documents. P@5 [16] is useful when users are primarily interested in the top results, as it focuses on the quality of the search system's most relevant suggestions. It is calculated as the number of relevant documents retrieved in the top 5 results divided by 5. P@5 can range from 0 (no relevant documents in the top 5) to 1 (all top 5 documents are relevant).

3. NDCG [17] (Normalized Discounted Cumulative Gain): NDCG is a measure that evaluates the ranking quality of a search system by taking into account the relevance of the documents and their positions in the ranked results. NDCG is particularly useful when the relevance of the documents is graded or weighted, instead of being binary (relevant or non-relevant). It is calculated by first computing the Discounted Cumulative Gain (DCG), which sums the relevance scores of the retrieved documents divided by the logarithm of their rank positions (discount factor). Then, the DCG is normalized by dividing it by the Ideal DCG (IDCG), which is the maximum possible DCG for a given query (i.e., when the results are ranked perfectly by relevance). NDCG ranges from 0 (worst) to 1 (best), with higher values indicating better search system performance.

These metrics are commonly used together to provide a comprehensive assessment of a search system or algorithm's performance in information retrieval tasks, considering different aspects of relevance and ranking quality.

On this data set using evaluation metrics used to compare the models MAP, P@5, and NDCG are as follows:

Method	MAP	P@5	NDCG
Cosine Similarity	0.0037	0.0080	0.0206
BM25	0.0030	0.0062	0.0191
Gensim	0.0037	0.0062	0.0207

Cosine similarity:

In this analysis, Cosine similarity achieved a MAP of 0.0037, P@5 of 0.0080, and NDCG of 0.0206. These values indicate that Cosine similarity performed slightly better than the BM25 model, but not as well as the Gensim model.

BM25:

The BM25 model achieved a MAP of 0.0030, P@5 of 0.0062, and NDCG of 0.0191. These values indicate that the BM25 model performed the worst among the three models tested.

Gensim:

In this analysis, Gensim achieved a MAP of 0.0037, P@5 of 0.0062, and NDCG of 0.0207. These values indicate that Gensim performed slightly better than Cosine similarity in terms of NDCG, but not as well as Cosine similarity in terms of P@5.

Overall, there is not a significant difference in performance between Cosine similarity and Gensim in terms of the evaluation metrics used. However, BM25 performed the worst among the three models tested.

CONCLUSION

Overall, the code worked well to perform the expected tasks of creating directories for the pre-processed files, deleting existing files, and pre-processing the data. The use of third-party libraries such as BeautifulSoup and NLTK makes the pre-processing tasks much easier and more efficient.

However, there are a few things that could be improved in the code in future. For example, some of the code uses hard-coded file paths, which could make it difficult to reuse the code on different systems or with different datasets. Additionally, some of the code could be refactored to make it more readable and modular. The code does not provide any analysis or evaluation of the processed data, so it is unclear how well the pre-processing and tokenization methods worked.

In terms of extending this work in the future, one possibility would be to incorporate additional pre-processing steps, such as part-of-speech tagging or named entity recognition, to further improve the quality of the pre-processed data. Additionally, the pre-processed data could be used to train machine learning models for information retrieval, such as deep neural networks or transformer-based models like BERT. The code could also be modified to include additional pre-processing steps, such as removing numbers or certain parts of speech. It could be improved by adding more pre-processing steps such as removing numerical data and special characters. The code could also be extended to include additional models for text analysis or search, such as deep learning models or clustering algorithms. Finally, the code could be used as a starting point for building a full-text search engine. The code could also be modified to support other datasets or information retrieval tasks.

REFERENCES

[1] Beautiful Soup, "Beautiful Soup: We called him Tortoise because he taught us," https://www.crummy.com/software/BeautifulSoup/bs4/doc

[2] Python Software Foundation, "Python 3.8.5 documentation," https://docs.python.org/3/

[3] NLTK Project, "Natural Language Toolkit," https://www.nltk.org/

[4] Gensim, "Topic Modelling for Humans," https://radimrehurek.com/gensim/

[5] C. D. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval, New York, NY: Cambridge University Press, 2008.

[6] R. Baeza-Yates and B. Ribeiro-Neto, Modern Information Retrieval. New York, NY, USA: ACM Press, 1999.

[7] P. G. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano, "To search or to crawl?: towards a query optimizer for text-centric tasks," in Proceedings of the 2001 ACM SIGMOD international conference on Management of data, 2001, pp. 265-276.

[8] R. A. Baeza-Yates, B. A. Ribeiro-Neto, and R. A. Baeza-Yates, Modern Information Retrieval. Pearson Education, 1999.

[9] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," Journal of machine Learning research, vol. 3, no. Jan, pp. 993-1022, 2003.

[10] T. Pedersen, S. Patwardhan, and J. Michelizzi, "Wordnet:: similarity: measuring the relatedness of concepts," in Demonstration papers at HLT-NAACL 2004, 2004, pp. 38-41.

[11] R. Rehurek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, 2010, pp. 45-50.

[12] C. D. Manning, P. Raghavan, and H. Schütze, Introduction to information retrieval. Cambridge University Press, 2008.

[13] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," Advances in neural information processing systems, vol. 26, pp. 3111-3119, 2013.

[14] C. D. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval. Cambridge University Press, 2008

[15] C. D. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval. Cambridge University Press, 2008.

[16] B. Carterette, "Evaluation in (almost) real time," in Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval - SIGIR '10, 2010.

[17] K. Järvelin and J. Kekäläinen, "IR evaluation methods for retrieving highly relevant documents," in Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '00, 2000.