Comparing IR Methods in Article Retrieval

Introduction:

Imagine that you are faced with a complex question or issue that requires expert knowledge to resolve. You might turn to scientific papers to find the answers you need, but with the vast amount of information available, it can be challenging to know where to start. This is where our Document Retrieval System comes in, a powerful tool designed to help you quickly and efficiently identify scientific papers that contain the answers you are looking for.

Our system uses a sophisticated approach to sentence-based retrieval. To achieve this, we first needed to understand how to create sentence embeddings. We utilized a range of traditional and modern techniques, including Tf-IDF, Word2Vec, and the Bert transformer, to transform sentences into embeddings. The core idea behind our retrieval system is to compare sentence embeddings using cosine similarity, a widely used method for measuring the similarity between two vectors.

While our Persian dataset is not particularly large, we still optimized our search process to ensure maximum efficiency. Rather than relying solely on a brute force approach of searching every sentence of every section of each paper, we employed a heuristic method to first identify the most similar papers based only on title, keywords, and abstract sections. We then searched the body_text section of those papers to identify the specific sentence that contains the answer to the user's question. This approach saved a significant amount of time and memory, particularly for large datasets.

Our Document Retrieval System is a powerful tool that can save researchers, scientists, and anyone seeking information a considerable amount of time and effort. By providing a quick and efficient way to identify relevant scientific papers, our system enables you to focus on what really matters: finding the answers you need to advance your research and knowledge.

DataSet:

Our dataset, stored in a JSON file, consists of four key components: title, keywords, abstract, and text body. It is worth noting that we have selectively chosen to include only the relevant portions of the text body, rather than the entire body.

Note:

We collected our own dataset, as we couldn't find a suitable dataset that met our needs. The dataset we used is available in two files: "dataset" contains only 12 articles, while "dataset1" contains almost 30 articles.

Code:

After importing and installing the library needed then we read the JSON dataset.
after that checking if a CUDA-enabled GPU is available and setting the device to run on the GPU if available, or on the CPU otherwise.
Then put the JSON to a list of dictionaries then tokenize each paragraph of abstracts, keywords and text body to a sentence.

Creating proper queries:
'متن کاوی آیا متن کاوي شامل مجموعه ابزار هاي هوشمندي است که براي سازماندهی اطلاعات بدون ساختار از آن استفاده میشود؟',
'رابطه انسان و فناوری چیست؟ در کجا استفاده میشود؟',

using “rtl_print” to print the persian font in a right to left form.

Brute Force approach:

In this approach we get similarity between queries and abstract, keywords and text bodies of the articles.

TF-ID:

This part of code is generating embeddings for a set of Persian language articles using the term frequency-inverse document frequency (tf-idf) weighting scheme. Here is a brief explanation of what each part of the code does:

initializes an empty list to hold the sentences of the articles then loop iterates over each article in the `data` list.

creates a `TfidfVectorizer` object, which is used to compute the tf-idf scores for the sentences.
Then fits the `TfidfVectorizer` object to the sentences, which creates a vocabulary of words and computes their corresponding tf-idf scores.

initializes an empty list to hold the tf-idf embeddings for each article.

- `article_sentences = []`: This initializes an empty list to hold the sentences of the current article.
- `sentence_vectors = vectorizer.transform(article_sentences)`: This transforms the sentences of the current article into a matrix of tf-idf scores using the previously fitted `TfidfVectorizer` object.

converts the tf-idf scores to a NumPy array then computes the weighted sum of the sentence vectors to obtain the embedding for the article.

- `tfidf_embeddings[i].append(article_embedding)`: This appends the article embedding to the list of embeddings for the current article.
- `tfidf_embeddings[i] = np.concatenate(tfidf_embeddings[i])`: This concatenates the embeddings for the current article into a single array.
After vectorizing the query given using tf-idf, we perform a similarity search between a set of query embeddings in tf-idf and a set of tf-idf embeddings for Persian language articles using cosine similarity.

- `for qindex, query in enumerate(query_embeddings):`: This loop iterates over each query embedding in the `query_embeddings` list, which contains the embeddings of the user's questions.
creates an empty priority queue, which will be used to store the most similar sentences for each query. iterates over each tf-idf embedding in the `tfidf_embeddings` list, which contains the embeddings of the sentences in the articles.

computes the cosine similarity between the current query embedding and the current tf-idf embedding.
- `max_sim_index = similarity.argmax()`: finds the index of the most similar sentence in the current article.
then adds the most similar sentence to the priority queue, along with its score and index information.
This retrieves the most similar sentence from the priority queue.

here is the result of specific queries and the related articles:

سوال: رابطه انسان و فناوری چیست؟

پاسخ 1: فناوری
از مقاله: پیش بینی جایگاه علم، فناوری، و نوآوری موسسه های آموزش عالی جهان در نظام رتبه بندی جهانی با استفاده از شبکه های عصبی مصنوعی
امتیاز: 0.3678043958202968

پاسخ 2: اما آیا اثر آموزش الکترونیک بر پایه فناوری اطلاعات چنین زیاد و تعیین کننده است؟ و اساساً فناوری و انسان چه نقش و نسبتی در این شیوه آموزش دارند؟ این از پرسش های بنیادی است که در خصوص ماهیت رابطه انسان/فناوری و آموزش/انسان/فناوری در آموزش الکترونیک وجود دارد و نیازمند بررسی است.
از مقاله: رابطه انسان و فناوري در آموزش الكترونيك از چشم انداز نظريه كنشگر شبكه
امتیاز: 0.3308122052207206

پاسخ 3: فناوری اطلاعات
از مقاله: پردازش نظام مند قانون؛ مطالعه موردی قانون پیشگیری و مقابله با تقلب در تهیه آثار علمی
امتیاز: 0.29093911334632355

پاسخ 4: این پایگاه دستاورد کاربرد فناوری اطلاعات برای مدیریت اطلاعات علم و فناوری در «پژوهشگاه علوم و فناوری اطلاعات ایران (ایرانداک)» است.
از مقاله: کاربست قوانین انجمنی و خوشه بندی در کنترل کیفیت داده های پژوهشی مورد مطالعه: پایگاه اطلاعات علمی ایران (گنج)
امتیاز: 0.1828943280056456

Cons of tf-idf in our project:
1) Inability to capture context: TF-IDF considers each term in the document independently and does not capture the context in which the term appears.
2) doesn't really good for the keywords as an answer, for example:
سوال: رابطه انسان و فناوری چیست؟
پاسخ 1: فناوری
از مقاله: پیش بینی جایگاه علم، فناوری، و نوآوری موسسه های آموزش عالی جهان در نظام رتبه بندی جهانی با استفاده از شبکه های عصبی مصنوعی
امتیاز: 0.3678043958202968

3) Sensitivity to document length: TF-IDF weights terms based on their frequency in the document, which can be problematic when comparing documents of different lengths. Longer documents tend to have higher raw term frequencies, which can bias the weighting towards common terms that may not be informative.

4) Limited semantic understanding: TF-IDF treats terms as discrete units and does not capture the semantic relationships between them.

Word2Vec:

Due to the limitations of traditional TF-IDF in capturing semantic relationships between terms, we have adopted a more modern approach that addresses this issue.
Word2vec is a popular unsupervised learning algorithm used for lots of tasks like document retrieval. It can be used in document retrieval to represent each document as a vector of word embeddings, which capture the semantic meaning of the words in the document.

We used **Hazm library**, Hazm is a natural language processing (NLP) library for the Persian language.
Tokenize each sentence of data(abstract, text body, keywords) into individual words in an array.
Generate word embeddings for each word using word2vec.
Average the word embeddings for all the words in the sentence to get a single embedding for the entire sentence.
article_sentences_words: This is a list of sentences, where each sentence is a list of words. This is the input corpus of text that will be used to train the word embedding model.

vector_size: This parameter sets the size of the word vectors (or embeddings) that will be generated by the model. In this case, the vectors will have a size of 300, which means that each word will be represented as a vector of 300 dimensions.

window: This parameter sets the maximum distance between the current word and the predicted word in a sentence. In other words, it determines the size of the context window that the model will use to generate word embeddings. In this case, the window size is set to 5, which means that the model will consider the 5 words to the left and the 5 words to the right of each target word.

min_count: This parameter sets the minimum frequency threshold for words to be included in the model. Words that occur less frequently than min_count times in the corpus will be ignored. In this case, the minimum count is set to 5, which means that any word that occurs less than 5 times in the corpus will be ignored.

workers: This parameter sets the number of worker threads to use for training the model. In this case, 4 worker threads will be used to speed up training.

Sentences embedding:

First, an empty list w2v_embeddings is created to store the embeddings for each sentence in the input data.

Then, for each document in the input data (represented as a list of sentences), the code loops over each sentence in the document.

For each sentence, the code initializes an empty list sentence_weights to store the embeddings for each word in the sentence, and an array sentence_embedding of zeros to accumulate the sum of all the word embeddings in the sentence.

The code then tokenizes the sentence using the word_tokenize function from the nltk library, and checks if each word in the sentence is in the word2vec model's vocabulary using the in operator.

If the word is in the model's vocabulary, the code retrieves the word embedding using the model.wv[word] method, adds it to the sentence_weights list, and accumulates it in the sentence_embedding array.

After processing all the words in the sentence, the code checks if there are any word embeddings in the sentence_weights list. If there are, it calculates the normalized weights for each word embedding using the L2-norm and stores them in the sentence_weights array.

Finally, the code calculates the weighted average of all the word embeddings in the sentence using the accumulated sentence_embedding array and the normalized sentence_weights array.

This gives the final sentence embedding, which is added to the sentences_embedding list for the current document.

Now it is time to vectorize the query:
First, an empty list query_embeddings is created to store the embeddings for each query in the input data.

Then, for each query in the input data, the code initializes an empty list query_weights to store the embeddings for each word in the query, and an array query_embedding of zeros to accumulate the sum of all the word embeddings in the query.

The code then tokenizes the query using the word_tokenize function from the nltk library, and checks if each word in the query is in the word2vec model's vocabulary using the in operator.
If the word is in the model's vocabulary, the code retrieves the word embedding using the model.wv[word] method, adds it to the query_weights list, and accumulates it in the query_embedding array.
After processing all the words in the query, the code checks if there are any word embeddings in the query_weights list. If there are, it calculates the normalized weights for each word embedding using the L2-norm and stores them in the query_weights array.

Finally, the code calculates the weighted average of all the word embeddings in the query using the accumulated query_embedding array and the normalized query_weights array.
This gives the final query embedding, which is added to the query_embeddings list for the entire data set.

After processing all the sentences in the current document, the sentences_embedding list is added to the w2v_embeddings list for the entire data set.

here is the results based on cosine similarity :

سوال: آيا متن كاوي شامل مجموعه ابزار هاي هوشمندي است كه براي سازماندهی اطلاعات بدون ساختار از آن استفاده ميشود؟

پاسخ 1: متن كاوي شامل مجموعه ابزار هاي هوشمندي است كه براي سازماندهی اطلاعات بدون ساختار از آن استفاده ميشود.
از مقاله: تحلیل احساس نظرات فیلم ها با استفاده از ماشین بردار پشتیبان دوقلو کمترین مربعات
امتیاز: 0.9734078899250955

پاسخ 2: قسمت عمده ای از آن چه که یک تحلیلگر خط مشی باید بداند، زبان محور است.
از مقاله: پردازش نظام مند قانون؛ مطالعه موردی قانون پیشگیری و مقابله با تقلب در تهیه آثارعلمی
امتیاز: 0.8808971981874189

پاسخ 3: وظیفه تحلیلگران خط مشی از بعضی جهات مانند مهندسان کامپیوتر است؛ اما از بعضی جهات فراتر از آن است و شامل در نظر آوری مولفه های پیچیده ای از جامعه، سیاست و اقتصادی نیز می شود.
از مقاله: پردازش نظام مند قانون؛ مطالعه موردی قانون پیشگیری و مقابله با تقلب در تهیه آثارعلمی
امتیاز: 0.8522085278672774

پاسخ 4: روش شناسی: پژوهش حاضر از نوع کاربردی است که با استفاده از روش مشاهده اجرا شده است.
از مقاله: کشف الگوهای کلیک کاربران برای استفاده در پرس و جو در پایگاه های اطلاعاتی
امتیاز: 0.8428671525524263

As you can see, we retrieved some sentences that don't have the specific terms on the query but retrieved the similar meaning and terms.
We got more similar articles compared to tf-idf.

Cons of W2V in our project:
1) Contextual information: Word2vec generates static embeddings that represent each word in isolation, without taking into account the context in which the word appears.
2) Out-of-vocabulary words: Word2vec is limited to generating embeddings for words that appear in the training corpus. Out-of-vocabulary (OOV) words, or words that do not appear in the training corpus, cannot be represented using word2vec embeddings.

Bert Transformer:

BERT (Bidirectional Encoder Representations from Transformers) is a powerful deep learning model that has been used in tasks, including document retrieval.
BERT is a pre-trained model that can be fine-tuned for specific downstream tasks, such as document retrieval.

So in our project In Article retrieving, BERT can be used to encode documents and queries into dense vectors that capture semantic meaning. These vectors can then be compared using cosine similarity to find the most relevant documents for a given query.
The cosine similarity between the document and query vectors is then used to rank the documents in order of relevance.
The model we used **"sentence-transformers/paraphrase-multilingual-mpnet-base-v2"** is based on BERT. Specifically, it is based on a multilingual version of BERT called "Multilingual-MLM" (Masked Language Model), which was pre-trained on a large corpus of text in multiple languages.
The "paraphrase-multilingual-mpnet-base-v2" model is a variant of the BERT model that is fine-tuned for the task of paraphrase detection. It is trained to encode pairs of sentences into dense vectors that capture their semantic similarity.

First, the code initializes a SentenceTransformer model using the paraphrase-multilingual-mpnet-base-v2 pre-trained model, which is designed to generate high-quality multilingual sentence embeddings.

Then, the code initializes two empty lists: first_sentence_embeddings to store the embeddings for each article in the input data, and flatten_data to store the flattened version of the article text (including both abstract and body text, as well as keyword phrases).

For each article in the input data, the code loops over each sentence in the flattened version of the article text, and adds it to the flatten_data list.
The code then uses the **SentenceTransformer model** to generate embeddings for the entire flatten_data list using the model.encode method. **The resulting embeddings are returned as a PyTorch tensor of shape (num_sentences, embedding_dim).**
The code then appends the resulting embeddings to the first_sentence_embeddings list for the current article, and concatenates them using the torch.cat method to create a single tensor of shape (num_sentences, embedding_dim) for the entire article.
After processing all the articles in the input data, the first_sentence_embeddings list contains a tensor of embeddings for each article, where each row corresponds to a sentence in the article.
Now it is time to vectorize the query with bet. And got a cosine similarity between the query and each sentence vector.

Now here is the result using Bert pretrained model to vectorize the query and sentences and then cosine similarity between them. It is obvious that these are more related.

سوال: آیا متن کاوي شامل مجموعه ابزار هاي هوشمندي است که براي سازماندهی اطلاعات بدون ساختار از آن استفاده میشود؟

پاسخ 1: متن کاوي شامل مجموعه ابزار هاي هوشمندي است که براي سازماندهی اطلاعات بدون ساختار از آن استفاده میشود.
از مقاله: تحلیل احساس نظرات فیلم ها با استفاده از ماشین بردار پشتیبان دوقلو کمترین مربعات
امتیاز: 0.9339420795440674

پاسخ 2: در واقع کاربر اطلاعات مفیدی از بررسی اولیه نتایج به دست می آورد.
از مقاله: کشف الگوهای کلیک کاربران برای استفاده در پرس و جو در پایگاه های اطلاعاتی
امتیاز: 0.5629725456237793

پاسخ 3: سوال مطرح در پژوهش حاضر این است که رفع ابهام از برچسب نحوی که فراوانی بالایی در پیکره های متنی فارسی دارند، چه تاثیری بر کارایی یک سیستم برچسب زنی خودکار (مطالعه موردی: سیستم برچسب دهی خودکار) دارد؟
از مقاله: بررسی امکان افزایش صحت یک ابزار برچسب دهی به اجزای کلام در فارسی

امتیاز: 0.5628782510757446
پاسخ 4: اطلاعات پژوهشی به مجموعه فراداده هایی گفته می شود که به فعالیت پژوهشی مربوط است.
از مقاله: کاربست قوانین انجمنی و خوشه بندی در کنترل کیفیت داده های پژوهشی مورد مطالعه: پایگاه اطلاعات علمی ایران (گنج)
امتیاز: 0.5616824626922607

Note: The first answer generated using BERT may be very similar to the original query, as it has been specifically trained to generate high-quality sentence embeddings that capture contextual information. However, the other answers may not have as high of a similarity score as word2vec, because BERT is able to understand the context in which each sentence appears, and these articles may be less directly related to the query in that context. While BERT can generate high-quality embeddings that capture the nuances of language, including sentence meaning and context, it may not always be able to generate embeddings that are as directly comparable to those generated by word2vec, **which is a simpler model that focuses more on capturing the semantic relationships between individual words.**

Conclusion:

TF-IDF is a simple and widely used method for information retrieval. It works by assigning weights to each term in a document based on its frequency and inverse document frequency. While TF-IDF can be effective for retrieval tasks, it does not capture the semantic relationships between words.
Word2Vec is a neural network-based method that can capture the semantic relationships between words by representing them as dense vectors in a high-dimensional space. This allows for more accurate retrieval of documents that contain similar or related concepts to the query. However, Word2Vec can be computationally expensive and requires a large amount of training data to generate accurate embeddings.

BERT (Bidirectional Encoder Representations from Transformers) is a state-of-the-art deep learning model that is pre-trained on large amounts of text data and can be fine-tuned for specific tasks such as document retrieval. BERT is highly context-aware and can capture the relationships between words and phrases within a document. It has demonstrated strong performance on a variety of natural language processing tasks, including information retrieval. In general, the choice of model for document retrieval depends on the specific requirements of the task at hand. If speed and simplicity are a priority, TF-IDF may be a good choice. If semantic relationships between words and phrases are important, Word2Vec may be a better option. For more complex retrieval tasks where context is critical, BERT may be the most effective model to use. Ultimately, it is important to consider the strengths and weaknesses of each model and choose the one that best fits the requirements of the specific problem.

After implementing the BERT transformer model, we found that it was significantly faster and more context-aware for document retrieval. In comparison to traditional methods, we also found that the Word2Vec approach produced more realistic results. We outlined the advantages and disadvantages of each method, and ultimately found that the BERT model performed better overall. However, we acknowledge that the choice of method ultimately depends on the specific situation and problem at hand.