Project overview

The tasks in this project can be split into 2 major parts:

1. Preprocessing the data: 1. Loading the data 2. Splitting the data into chunks 3. Generating vector embeddings 4. Saving the preprocess dataset into a file
2. Indexing and retrieval 1. Loading the preprocessed data 2. Indexing the vector embeddings 3. Generating the vector embedding for the target query 4. Retrieving the relevant results (5 in this case)

Tasks

Loading and saving the data

I used the `pandas` library due to the built in `read_csv` and `to_csv`. Moreover the `DataFrame` stucture allowed me to easily analyze the content as well as convert between different data structures depending on the needs.

	Title	Text
count	1391	1391
unique	1390	1391
top	Autonomous Agents And Multi-Agent Systems 101:...	1. Introduction of Word2vec\r\n\r\nWord2vec is...
freq	2	1

From the simple description of the `DataFrame` it became clear that the title cannot be used as an identifier and that the Text contains unnecessary characters -> `\n \r`.

Formating the data

I decided to add an additional column 'ID' so that the articles can be clearly distinguished. (The 2 articles with the same title appear to be almost the same with minor differences such as the date).

I decided to split the texts into sentences, because they contain a complete piece of information. To do that I used the `Sentencizer` from `spaCy`.

Having obtained the texts cleaned up and split into sentences I procceded to split into chunks of 10 sentences (initialy I tried 5 sentences - a common number of sentences in a paragraph but I increased it to 10 when it didn't visibly impact the efficiency)

	ID	chunk_char_count	chunk_word_count	chunk_token_count
count	6263.00	6263.00	6263.00	6263.00
mean	697.83	1222.13	194.76	305.53
std	401.58	567.78	82.53	141.95
min	0.00	3.00	1.00	0.75
25%	345.00	909.00	150.00	227.25
50%	714.00	1192.00	194.00	298.00
75%	1053.50	1498.50	239.00	374.62
max	1390.00	7942.00	883.00	1985.50

Creating the vector embeddings

The model I decided on to create the vector embeddings is `BAAI/bge-base-en-v1.5`

I chose it based on Retrieval Average Score in the leaderboard: https://huggingface.co/spaces/mteb/leaderboard and the small size of the model (0.41 GB).

This model can handle up to 512 tokens. While it's sufficient for most of the chunks there is at least one where we lose some data. Notably if we use 5 sentences per chunk the max `chunk_token_count` is ~1400. This is a potential area of improvement - we can use a model that can handle more tokens or revise the way we chunk the articles for example: forcibly split sentences that are too long.

Indexing the embeddings

The first real challenge was picking an appropriate algorithm for indexing the vector embeddings. The first one I came across was `faiss` but only the CPU version is availible on Windows. I had the same issue with `ScaNN`. Finally I settled on `hnswlib`

https://github.com/nmslib/hnswlib?tab=readme-ov-file

While works on CPU it managed to achieve good results on ANN Benchmarks https://ann-benchmarks.com/

For the 'distance' I chose cosine similarity since it is often used to measure document similarity in text analysis.

Retrieval

I computed the vector embedding of the query and used the `knn_query` from `hnswlib` to retrieve the relevant records.

Areas for improvement

If the dataset where to change, especially the size of it, it might be necessary to use a vector database. The current solution works well largely due to the small amount of data.

Another thing to consider is the operation system of the target device, if it's Linux it might be a good idea to check whether the algorithms such as `ScaNN` don't perform better on our data.

Another thing to improve would be expanding the functionality to handle multiple queries at the same time (likely switching to an algorithm that uses the GPU for calculations).