Project - Carlos Segura

Resources

The Document Collections is in Wikipedia1.6M
The Query Log is in AOL Query Logs

Statistics on DC

There are 1,662,756 documents in the document collection.
After we applied the text processing techniques, we got an index of 91,979 terms.
We needed to apply general stop word removal techniques and more custom techniques based on our document collection.

Text Processing

We decided to use the Stopword list from https://www.ranks.nl/stopwords because it's very general to use. We added the , (comma), . (dot) and : (colon) to the stop word list to reduce the amount of token from those characters . We decided to store words of length greater than 2 in the inverted index because those small-size words don't usually add value to a query.

We use the word tokenizer of NLTK because it's a standard word tokenizer and pretty used in the industry.

We use the PorterStemmer from https://www.nltk.org/. We decided to use the Porter Stemmer because it is simple and fast.

There are some special characters that appear at the beginning or end of some words in the index which are removed, characters such as ', -, ., =, \
Numbers are not included in the index because we think they were difficult to make them clean and they do not add so much value to most queries. That's why we use the NLTK corpus words as a dictionary to filter the words to only get Enligsh words and disregard other languages. Our search engine works only on English at this stage.

We use MongoDB as our data storage for the inverted index, document collection and pre-processed document information. We decided to use MongoDB because it is an unstructured database, which gives us flexibility for the future, but most importantly because it is very efficient in read operations, which are the predominant operations in a search engine. The theory tells us that the inverted index needs to be stored alphabetically, so we created the inverted index that way, but additionally we added a MongoDB index in each term to make the read operations even faster.
A search engine must not modify the document's content, so we are not doing any formating or modification to any document in our collection. We are just saving them in MongoDB as they are.

We struggled many times to create the inverted index, more information about this in the Discussion section, but at the end we needed to use multiprocessing and batch programming to create it successfully. We processed the wikipedia_text_files.csv as chunks of data of 10,000 documents per batch. We created multiple processes in each batch to get the index terms of the documents in parallel. We created pickle files as an intermediate storage for the index. After all documents are processed and saved in MongoDB, then we iterate through the pickles files to create a unified inverted index, which then is saved in MongoDB.

Query Suggestions

We relied on Pandas Dataframes to manipulate the logs files from merging them to one file to get the query suggestions. We chose Pandas Dataframe because it is very performant on the type of operations we needed to execute on arrays.

Relevance Ranking

We relied on the data we have on MongoDB (index term, maximum term frequency in the document) and the use of python list and dictionary to create the algorithm for relevant ranking.

Snippet Generation

We created the algorithm from scratch without using any library. We used standard python lists and dictionaries.

UI

Our UI is a Web UI, we chose Flask as a web back-end framework and jquery as a UI library manipulation. The UI is very simple, consisting of only 1 page. We use ajax technology for communication between the UI and the server.

Lessons Learned

The most complicated challenge was to deal with a 5gb file with more than 1.6 million documents inside. We tried several strategies such as loading all documents in memory, reading the file all at once, create a data structure to store all documents in memory using redis, and other strategies. We failed on most of them due to the huge amount of data we needed to store on memory to apply all the operations. After many failed attempts, we realized that it would be impossible to continue with that strategy, so we changed the strategy to store everything in a fast document store such as MongoDB. We needed to use multiprocessing to reduce considerably the amount of time needed to process all documents and create the data structures and populate them using the algorithms.

Also, we needed to be more restrictive in the text processing to make sure our inverted index does not store “garbage terms”, which are uncorrected words. That’s why we applied all the techniques explained in the section Text Processing.

It is clear that Pandas Dataframes really make it easy to work with vectors. Also, they are really performant compared to doing it in the python way.