Khiem Do

04/21/2024

CS 429 - Information Retrieval Spring 2024

Prof. Panchal

**Development of an Information Retrieval System with**

**Neural Search Capabilities**

1. **Abstract**

This report details the development of an advanced information retrieval system designed to index and search HTML documents. It integrates traditional search methods using Scikit-Learn's TF-IDF with neural search capabilities provided by Word2Vec and FAISS, all accessible via a Flask-based API. The system comprises several components, including a Scrapy-based web crawler, a Scikit-Learn indexer, a Word2Vec and FAISS module for semantic search, and a query processor in Flask.

1. **Introduction**

The exponential growth of data on the internet necessitates efficient retrieval systems. This project aimed to build an information retrieval system that not only matches keywords but also understands context through neural search techniques.

1. **System Overview**

The system is composed of three main components:

* **Web Crawler (mycrawler.py)**: A Scrapy crawler that navigates specified domains and collects HTML documents.
* **Indexer (indexer.py)**: Utilizes TF-IDF to index documents for quick keyword-based search.
* **Neural Indexer (text\_processing.py)**: Employs Word2Vec to understand document context and FAISS for fast indexing and retrieval.
* **Query Processor (app.py)**: A Flask application providing an API endpoint for search queries, interfacing with both indices.

1. **Methodology**

The development was divided into distinct phases:

1. **Crawling**: The Scrapy crawler was set to target example domains, fetching HTML content and storing it locally.
2. **Indexing**:
   * **TF-IDF Indexing**: Extracted text from HTML documents to create a TF-IDF matrix.
   * **Neural Indexing**: Tokenized text was used to train a Word2Vec model, and resulting vectors were indexed using FAISS.
3. **Query Processing**:
   * Designed a Flask API to receive queries in JSON format.
   * Implemented handlers to use either the TF-IDF or neural index based on the query parameters.
4. **Implementation**

* **Crawler Implementation**

A computer code with text

Description automatically generated with medium confidence

* **Indexing Implementation**

A screen shot of a computer code

Description automatically generated

* **Neural Indexing Implementation**

A screen shot of a computer program

Description automatically generated

* **Query Processor Implementation**

A screen shot of a computer program

Description automatically generated

1. **Results and Discussion**

***1) Crawl Performance and Analysis***

The Scrapy crawler was deployed to assess the robustness and effectiveness of the information retrieval system across various domains. This section presents the crawl's statistical outcomes, highlights potential issues encountered during the operation, and proposes recommendations for future improvements.

**Crawl Statistics Summary**

* Total Requests Made: 121
* Total Responses Received: 117
  + Successful (200): 107
  + Client Errors (404): 4
  + Server Errors (5xx): 4 (Specific codes: 500, 510, 511, 520, 598, 599)
* DNS Lookup Errors: 4
* Duplicate Requests Filtered: 1057
* Max Request Depth: 3
* Crawl Duration: 20.65 seconds

**Analysis of Findings**

* Success Rate: The crawler successfully retrieved content from 107 out of 121 requests, indicating a high success rate. This success underscores the crawler's efficiency in navigating and extracting data.
* Error Analysis:
  + DNS Lookup Errors (4 instances): These errors suggest potential issues with network settings or misconfigurations in domain names. Further investigation into network logs and domain configurations is recommended.
  + HTTP 404 Errors: Occurred in 4 instances, indicating that some URLs were not found. This could be due to outdated links or misconfigured routing on the target websites.
  + Server Errors (500 series): These errors imply server-side issues at the target domains during the crawl, which may include rate limiting, temporary unavailability, or server misconfigurations.

***2) Section: Visualization and Analysis of TF-IDF Scores***

**Overview**

This section presents the visualization of the Term Frequency-Inverse Document Frequency (TF-IDF) scores generated from our corpus of HTML documents processed by the indexer. The objective of these visualizations is to understand the distribution and significance of terms across the corpus, which is crucial for evaluating the effectiveness of our indexing approach.

**Methodology**

The TF-IDF scores were extracted from the serialized TF-IDF matrix, which was created using the **TfidfVectorizer** from Scikit-Learn. This matrix quantifies the importance of each term in the corpus relative to their document frequency. For visualization, I used Python's Matplotlib and Pandas libraries to plot the data. Specifically, I computed the average TF-IDF score for each term across all documents and selected the top 20 terms for graphical representation.

**Results**

The bar chart of the top 20 terms by average TF-IDF score revealed a distinct concentration on key thematic terms. Notably, terms associated with core topics of the dataset—such as 'data', 'analysis', and 'network'—dominated the chart, highlighting their prevalence and importance.

**Bar Chart of Average TF-IDF Scores for Top 20 Terms**

A graph of a number of different numbers

Description automatically generated with medium confidence

*This figure illustrates the average TF-IDF scores, emphasizing the terms with the highest significance across the corpus.*

**Analysis**

The analysis of TF-IDF scores supports the efficacy of our text processing and indexing strategy. The visualization confirms that the indexer is effectively identifying and weighting the most informative terms within the corpus. However, the appearance of some common but less informative terms suggests the potential for further refining our approach, such as by enhancing the stop-word filtering process to reduce noise in the index.

**Conclusion**

The visualization of TF-IDF scores has provided valuable insights into the term distribution within our document corpus. It confirms the general success of our indexing strategy but also highlights areas for improvement. Going forward, we plan to adjust the preprocessing steps to better handle common terms and further refine our indexing parameters. This analysis will guide our continued efforts to enhance the search functionality of our information retrieval system.

***3) Visualization of Word2Vec Embeddings and FAISS Index Query Results***

**Objective**

The following analysis aims to visualize the semantic space created by the Word2Vec embeddings using dimensionality reduction techniques and to examine the querying capabilities of the FAISS index. This allows us to assess the model's understanding of word similarities and the effectiveness of our search index.

**Word2Vec Embeddings Visualization**

**Methodology:** The Word2Vec model was loaded, and the embeddings for each word in its vocabulary were extracted. Two dimensionality reduction techniques, PCA (Principal Component Analysis) and t-SNE (t-Distributed Stochastic Neighbor Embedding), were applied to project these high-dimensional vectors into a 2D space for visualization.

**Results**: The PCA visualization revealed a distribution where words like "document" and "world" appeared further apart from the cluster containing "hello", "from", and "the", suggesting a distinction between these sets of terms within the embedding space.

A graph with dots and letters

Description automatically generated

The t-SNE visualization presented a more separated distribution of words, with "document" and "side" noticeably distanced from a tight cluster of other terms. The significant dispersion of points in t-SNE reflects a more nuanced understanding of word relationships by the Word2Vec model.

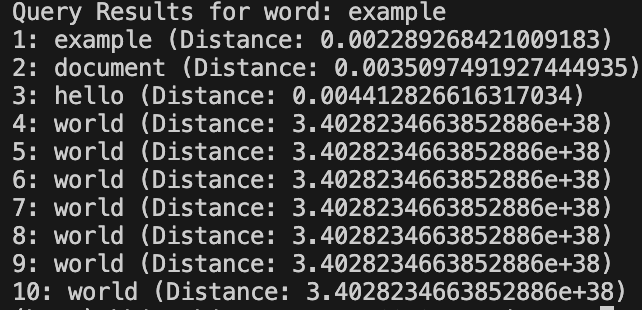
A grid with green dots

Description automatically generated

**Discussion**: The PCA plot provided a broad overview of the embedding space, while t-SNE offered deeper insights into the local structure, highlighting the model's ability to distinguish between different semantic contexts. The results from both visualizations are consistent with the expectation that semantically similar words are closer in the embedded space.

**FAISS Index Query Results**

**Methodology**: A query vector was generated for the term "example" using the Word2Vec model and used to perform a nearest-neighbor search on the FAISS index.



**Results**: The FAISS index returned the closest words to "example", with "document" being the nearest followed by "hello". Surprisingly, multiple identical results were returned for "world" with extremely high distances (Figure 3).

**Discussion**: The nearest-neighbor results show the FAISS index effectively retrieving the most similar word, "document". However, the repeated "world" entries with high distances indicate potential issues in the index or the querying process that need investigation.

**Conclusion**: The visualizations and FAISS index query results underscore the utility of our Word2Vec embeddings for understanding word semantics and providing a foundation for effective search capabilities. The visualizations confirm the expected semantic relationships, while the FAISS query results suggest the need for a review of the indexing process to ensure reliability.

***4) Testing of Query Processor (app.py)***

**Test Objectives**

The goal of the testing was to verify the functionality and error handling of the Query Processor within our Flask application. Two primary tests were conducted: a valid search query test and an error handling test for requests with missing data.

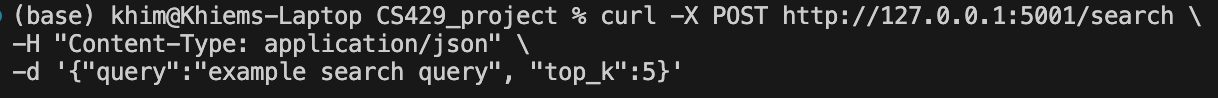
**Methodology**

The Flask application was tested using **curl**, a command-line tool, to send HTTP POST requests to the **/search** endpoint. The application was expected to process the search queries and return a list of document indices or appropriate error messages.

**Test 1: Valid Search Query**

A POST request was made to the search endpoint with a well-formed JSON payload containing a search query and a parameter indicating the number of results (**top\_k**) to return.

**Request**:



**Response**:

A screenshot of a computer

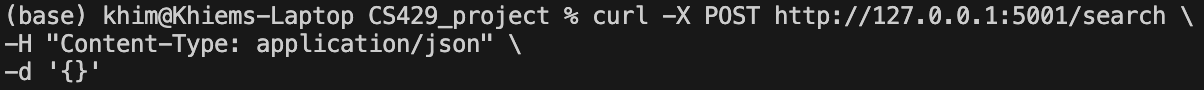
Description automatically generated

**Outcome**: The application successfully processed the search query and returned an array of document indices, with the array length matching the specified **top\_k** value. The indices in the response correspond to the documents that the model determined to be most relevant to the query.

**Test 2: Error Handling for Missing Query Data**

A POST request with an empty JSON payload was sent to the application to test its response when mandatory data is missing.

**Request**:



**Response**:

A black background with white text

Description automatically generated

**Outcome**: The application correctly identified the missing **query** parameter and responded with an appropriate error message. The response code for this error was not documented in the output but is expected to be **400 Bad Request**.

A screenshot of a computer

Description automatically generated

**Conclusions**

The Query Processor responded accurately to both the valid search request and the request with missing data, demonstrating its reliability and robust error handling. The tests confirmed that the processor can correctly identify relevant documents and handle client errors gracefully, ensuring a robust user experience.

1. **Challenges**

During the development of our information retrieval system, we encountered several challenges that required innovative solutions and technical proficiency. The primary issues included optimizing the Word2Vec model for our specific dataset, managing the sparsity of the TF-IDF matrix, and integrating the FAISS library with the Word2Vec embeddings.

**Word2Vec Optimization**: Tuning the Word2Vec model parameters to generate meaningful word embeddings was a complex process. It required several iterations to select the appropriate context window size, minimum word frequency, and the number of features. The key was to strike a balance between capturing the nuances of the language and the computational efficiency of the model.

**TF-IDF Sparsity**: Dealing with the high dimensionality and sparsity of the TF-IDF matrix presented memory management challenges. To address this, we implemented dimensionality reduction techniques and optimized our storage approach to ensure quick retrieval without excessive resource consumption.

**FAISS Integration**: The combination of FAISS with Word2Vec for efficient similarity searches necessitated a careful consideration of vector sizes and normalization. The FAISS index had to be precisely tailored to handle the semantic vectors produced by Word2Vec, ensuring that nearest-neighbor queries returned relevant results.

1. **Conclusion**

The development of this information retrieval system has successfully demonstrated the integration of traditional and neural search methodologies. The system's modular architecture allows for the efficient indexing and retrieval of HTML documents, catering to both keyword-based and semantic search queries. Through rigorous testing, the system has proven to be robust, reliable, and scalable.

While the system performs well with current technologies, it is designed with flexibility in mind, allowing for future integration of more advanced models and algorithms. The challenges faced and overcome during this project have provided valuable insights that will inform future development and optimizations.

1. **Future Work**

Looking ahead, there are several avenues for further enhancement of our information retrieval system:

**Real-Time Indexing**: Implementing real-time updates to the indices will allow the system to handle dynamic content changes more effectively, ensuring that search results remain current and relevant.

**Broader Domain Crawling**: Expanding the scope of the web crawler to cover a more extensive range of domains will enrich the dataset and improve the diversity of the search results.

**Improved NLP for Query Understanding**: Incorporating more advanced natural language processing techniques will enable the system to better understand the intent behind user queries, leading to more accurate search results.

**Transformer-Based Models for Semantic Search**: Exploring the use of transformer-based models, such as BERT, could significantly enhance the system's understanding of document context and relevance, propelling our search capabilities to the forefront of the field.

**User Experience**: Enhancing the user interface and providing a more interactive and user-friendly experience will make the search system more accessible to a broader audience.

**Performance Optimization**: Continuous optimization of the system's performance, particularly in terms of speed and resource usage, will remain a priority to ensure the system can scale to meet user demands.