Oğuzhan Güngör
24411006

INF 528 ADVANCED TOPICS IN COMPUTER ENGINEERING
FINAL PROJECT REPORT

PART I – Administrative Features and Management Aspects

Project Title

Building a Knowledge Graph enriched with Large Language Models and Linked to DBpedia.

Team

Oğuzhan Güngör – Presenter

Oğuzhan Güngör – Data preparation & data preprocessing

Oğuzhan Güngör – Knowledge graph design and enrichment

Github

https://github.com/theFellandes/LLM-Final-Project

PART II – The Study

Abstract

The goal of this project is to enrich knowledge graph using DBpedia and LLMs. Key finding is that 2 million data is a huge data and handling it requires alternative approaches. Utilizing DBpedia and LLMs are good way to ensure data quality.

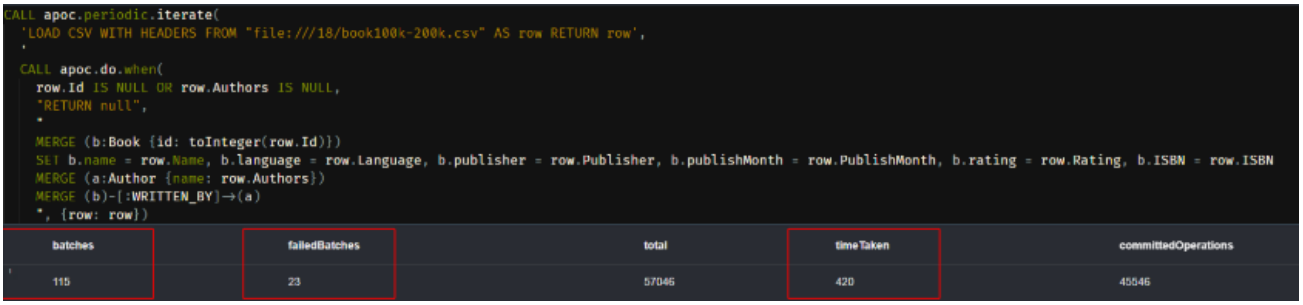
Introduction

This section aims to address the problem or challenge at hand, state the project's objectives. In this project the goal was create a knowledge graph using the **Goodreads Book Datasets with User Rating 2M**[i]. With this dataset and **DBpedia**[ii] and LLMs the goal is to create a knowledge graph that generates book suggestions in a clever way and observe which LLM performs better in comparison to others.

Materials and Methods

This section describes the tools, technologies, and methodologies used in the project's design and implementation. For the baseline of the project, Python and Neo4j has been used. The backend, data manipulation, querying all made possible using Python. Utilizing a Docker container containing Neo4j has been the solution for the database storage and initialization. Downloading the dataset made possible with Python. Using Kagglehub, the dataset was downloaded and stored into objects to be sent to Neo4j. After that, the objects have been stored inside the Neo4j database. For storing nodes, it was also possible to utilize APOC plugin however due to race condition, APOC plugin returned errors therefore utilizing a multiprocess approach was more suitable for the data integrity.

The results of APOC plugin have been shown below:



As shown in the screenshot captured from Neo4j's user interface (UI), there are failed batches, these were due to the concurrency used in the APOC process and 23 batches couldn't get the required lock for writing to the database. Therefore, they are discarded.

Then APOC plugin's script has been changed for better data handling and easier transferring however this also had data integrity issues but 0 issue approach was time costly therefore abandoned. Also, for improving the knowledge graph, utilizing DBpedia's database was tinkered with. Utilizing DBpedia for obtaining Genre and Awards improved database's quality. Since books were lacking the information about genre, descriptions and awards, they were added using DBpedia's database. Descriptions and genre were a requirement for similarity analysis with LLMs because using only book names was prone to errors. That's why genre and description were pulled from DBpedia.

For utilizing DBpedia, dbpedia_awards.py, dbpedia_description.py, and dbpedia_integration2.py were used and for utilizing those scripts, SPARQLWrapper, neo4j libraries were used. SPARQLWrapper was used because to communicate with DBpedia, neo4j was used for connecting to neo4j and adding data to knowledge graph. For these scripts, csv was used because sending a request and getting data from neo4j was cumbersome and writing from csv was faster compared to reading from database.

For sentiment analysis, LLMs were used and compared with each other. Mainly BART and OpenAI were used for sentiment analysis. Utilizing LangChain for LLM maintenance and interchanging between LLMs was the used method for multiple LLM comparison. In order to perform the sentiment analysis to support partial matching, regex was used and re was imported. This was implemented in order to make a partial match with the book and search the knowledge graph based on that partial match. If partial match returns 2 results, the first matching was returned. If it's not the desired match, the user can enter whole name.

For similarity analysis, firstly whole database was tried. However, this approach took so much time and that wasn't suitable. Therefore utilizing llm_similarity_langchain_1000.py was created to get random 1000 books from the knowledge graph and perform the similarity analysis on those books using OpenAI.

Tensorflow, langchain.embeddings, concurrent.futures were used for this part of the project. Tensorflow had used because of utilizing another LLM to perform the similarity analysis. Embeddings was used for sending batch by batch the books in order to perform the similarity analysis. Futures was used for speed, in order to utilize threads.

Results

This section presents the findings. The project was hard at the creating the knowledge graph part. Because it took too much time to store the data onto Neo4j. Many methods were tested in this stage of the project, the data processing part. One of the tried methods was running csv through a Python script and then insert it into Neo4j. This was time-consuming however it was the most effective in the case of accuracy because due to synchronized approach, it ensured the connection was never lost with the Neo4j server. The other method was to use the Neo4j's own import tool. This was faster than the previous method however it was not as accurate as the previous method. The reason for this was

the connection between the Neo4j server and the import tool was not as synchronized as the previous method. This caused some data loss in the process. Due to utilizing asynchronous method using APOC library, the data loss was huge. Another method was to apply concurrency with Python. This had the connection pooling issues with local Neo4j server. There were so many race condition errors and connection errors, I've abandoned this method. Tried methods can be found under the legacy folder. pooling&graphql contains the concurrency script as well as connecting to DBpedia. Making sentiment analysis was the easiest part because LangChain library was very easy to use. The only problem was the time it took to process the data. The sentiment analysis took too much time to process the data. The similarity analysis was not implemented due to the time constraints and issues that had arisen could not solve in the time frame. Using DBpedia and enriching the knowledge graph with it was also one of the hardest parts because the querying was very different from writing SQL queries. It required a lot of time to understand the querying language and how to use it. The other problem was the data was not as accurate as the data in the Kaggle dataset. This caused some data loss in the process. Utilizing multiprocessing instead of concurrency prove easier to handle due to connection pooling issues that arises with dockerized Neo4j. Therefore, storing whole database took too much time, around 16 hours. This time sink was also due to host computer's SSD having less storage capacity.

In the sentiment analysis part, BART and OpenAI models were used and their results were similar to each other. When it comes to positive or negative, both performed really good. Alas, OpenAI model was better when it comes to scoring 1 to 5. The results can be seen in the below image, when OpenAI model stating a review is 4 stars "really liked it", BART states that review to be 2 stars review. Even though BART understood this review is positive, it failed to acknowledge its stars.

```
Chatbot:
Reviews for any book name containing 'Gatsby':
  - User ID 4:41a437e1-0c7a-47fe-b54b-212d070d458a:585786:
    Review: "liked it"
    [OpenAI] => POSITIVE (4 stars)
    [BART]   => POSITIVE (4 stars)
  - User ID 4:41a437e1-0c7a-47fe-b54b-212d070d458a:585786:
    Review: "it was amazing"
    [OpenAI] => POSITIVE (5 stars)
    [BART]   => POSITIVE (4 stars)
  - User ID 4:41a437e1-0c7a-47fe-b54b-212d070d458a:585786:
    Review: "really liked it"
    [OpenAI] => POSITIVE (4 stars)
    [BART]   => POSITIVE (2 stars)
```

After performing similarity analysis for the entire dataset, it took too much time, therefore, utilizing a random 1000 books was performed. For this task utilizing OpenAI had some issues because it tried to fit in each given books, since dataset is too small, it failed to match other books and made some confusing matchings. For this similarity analysis, embeddings were generated and supplied to the LLMs. In figure below, it shows how much it took to process 79105 books.

```
   [DEBUG] Generating OpenAI embedding...
   [DEBUG] OpenAI embedding done in 0.35 seconds.
   [DEBUG] Generating Flan embedding...
   [DEBUG] Flan embedding done in 0.10 seconds.
   [DEBUG] Generating TF embedding...
   [DEBUG] TF embedding done in 0.00 seconds.
[DEBUG] Processing book 79105/79106 - ID: 5854
   [DEBUG] Generating OpenAI embedding...
   [DEBUG] OpenAI embedding done in 0.23 seconds.
   [DEBUG] Generating Flan embedding...
   [DEBUG] Flan embedding done in 0.36 seconds.
   [DEBUG] Generating TF embedding...
   [DEBUG] TF embedding done in 0.00 seconds.
[DEBUG] Processing book 79106/79106 - ID: 5860
   [DEBUG] Generating OpenAI embedding...
   [DEBUG] OpenAI embedding done in 0.23 seconds.
   [DEBUG] Generating Flan embedding...
   [DEBUG] Flan embedding done in 0.12 seconds.
   [DEBUG] Generating TF embedding...
   [DEBUG] TF embedding done in 0.00 seconds.
[INFO] All embeddings generated in 33436.36 seconds.
[INFO] Computing pairwise similarities...
```

Conclusion

Utilizing knowledge graph with Sparql and LLMs improves knowledge graph's knowledge significantly. Performing sentiment analysis using BART and OpenAI shows that in case of good or bad, both performs similarly, however in the case of 1 to 5, OpenAI performs significantly better.

References

i) https://www.kaggle.com/datasets/bahramjannesarr/goodreads-book-datasets-10m/data

ii) https://www.dbpedia.org/