Project Report

Soojung Choi, Tejas Sujit Bharambe

● Goal & Research Question

We built a knowledge graph about international trade between the US and all the countries in the world. KG in this domain was necessary as the US is connected with the world in the most complex way, and world trade is happening 24/7; it generates a lot of data that is messy and unstructured. Also, it is hard to connect and synthesize the trade data with country information, such as GDP, population, or if the country is FTA in force with the US. Hence, we needed a smart way to organize and manipulate the data to get answers.

This knowledge graph has primary entities like country, currency, item *(ex. meat of bovine animals)*, product sub-category *(eg. meat),* and product main section *(eg. animal products)*. The entities have primary information like the trading year, import value, export value, and some supporting information like bilateral FTAs between countries, GDP, population, etc.

Primarily, this KG lets users explore yearly trade information broken down at either the product, product-country, or country-country level. We have also added some interesting elements like a prediction of near-future trade between 2 countries, the recommendation system for the bilateral Free Trade Agreement (FTA) between countries, and a Q&A module to answer world trade-related questions.

● Datasets

This KG will contain the data crawled from the following sources:

⬡ https://oec.world/ *([World Trade Data - Trade, Countries, Products, Categories, Sections], unstructured, format - charts, text & links, ~35M records)*

⬡ https://data.worldbank.org/ *([GDP, Population], structured, format - tabular, ~13.5k records)*

⬡ www.state.gov/trade-agreements/existing-u-s-trade-agreements/ *(Free Trade Agreement Data, structured, format - tabular, 20-30 records)*

⬡ https://country-code.cl/ *(Country Codes, structured, format - tabular, ~250 records)*

● Approach

We crawled the data from the above-mentioned sources and to generate the triples in the desired format, certain steps like data cleaning, information extraction, and entity linking were performed. These triples were loaded into Neo4j aruaDB via the neo4j python driver to generate the knowledge graph. We then created a web app to allow users to interact with this knowledge graph in the form of visualizations, question-and-answering modules, and a recommendation system.

● Lessons with Technical Challenges (1.The process of building KG DB / 2.Data size and quality)

⬡ **Scraping data:** To extract data from the tabular data sources, we used the beautiful soup and for the charts, links, and textual data on the OEC world trade portal, we used Scrapy to crawl the data. Initially, a lot of our records were not getting extracted as the data in visualization took a longer time to load into the browser as compared to other components of the website. We effectively used the logging technique of Scrapy to identify this page loading issue and took measures such as wait time. Also, there were 35M records that were extracted from these pages, and in order to evaluate the correctness of this data, we pulled the summary numbers from the website itself and compared the sum totals. Additionally, we evaluated them by using spot checks and efficient exception handling.

⬡ **Information Extraction** The text/link data needed further processing to extract the exact information that we needed. We used rule-based information extraction mechanisms with a combination of lexical and syntactical rules. We leveraged spaCy's tokenizer, parser, named entity recognition, part-of-speech tagging, and dependency parsing heavily to form the rules and extract information like ranks, ECI, etc. This component was used in 2 places - one for information retrieval from data and one for the entity,

dependency noun and verbs extraction for the question-answer module. One challenge was to manually identify the type of patterns and keywords in the sentences. And assessing the quality of the information extraction was a major challenge & we achieved it by continuously adding, and enhancing our rules, and checking if the extracted data matched accurately to a set of pre-determined patterns.

⬡ **Entity Linking** (Similarity Measures): For building KG, country-related information was common in all the extracted files. We linked the country names using a blocking technique based on the first 2 characters and used Jaro and Dice similarity measures for entity linking after heavily testing with the available similarity measures. We created a truth file for this and got an accuracy score of 97.3%. We realized that the alternate country names were causing the issue. For eg., South Korea is called the Republic of Korea, and North Korea is the Democratic People's Republic of Korea. Hence, we referred to the alternate country names page on Wikipedia and made our training data more robust as the countries were critical for our KG. Additionally, along with country names, we used Jaro similarity to link the product names and year with the entities in KG from the questions that the users can ask.

⬡ **Ontology** - Designing an efficient ontology was of utmost importance as we had more than 35M records. We ensured that we didn't duplicate the country-country combinations by adding a Trade ID to each of the trade data. Similarly, we added the FTA id for FTA records too. Initially, the ontology was too complicated and had a lot of repetitive data but we simplified it by redesigning it and adding properties to the nodes.

⬡ **KG Quality assessment:** To ensure the high quality of the KG, we used a manual approach with a mix of subjective and objective criteria to verify the functional and non-functional requirements of KG. We cross-verified the counts of each relation, relation type, label, property, and node from KG with that of the raw data & also checked the sum totals for each country and product-based combinations.

⬡ **Data Visualization** We fetched the data from the Neo4j database with 70k nodes and 270 relationships using Cypher queries and Python neo4j driver. We built a visualization page using the Plotly library to show the data at the country level, such as GDP comparison, import/ export value comparison, yearly import/export value between countries, and the value by trade categories, and at the product level, such as yearly product trade value and section-category-item tree.

⬡ **Question and Answering:** We used a template-based question-answering module. As mentioned above, we performed information extraction and entity linking on the questions that users asked to extract and link country, products, year, and verbs. The question was then updated based on the linked entities from the KG. Based on that, the context was generated which was to be used for answering the questions with the scores. We ended up using the "distilbert-base-cased-distilled-squad" algorithm with the help of the hugging face library. We tested various other algorithms like "roberta-base-squad2", google's "tapas-base-finetuned-wtq". However, after much trial and testing of the questions, we realized the distilbert algorithm gives more accurate answers based on the probability scores generated using a softmax from the start and end logits of the matched token.

⬡ **Prediction & Recommendation** For prediction, we asked questions like how the export and import trade values would have been different if the US had FTA in force with a certain country years ago. And what will be the trade value like in the near future? To answer the questions, we trained models using scikit-learn library. We tried Linear Regression, Lasso, Ridge, SVM, and Random Forest models. Though it was challenging that the dataset is too enormous to scrape and build the database, it turned out it was too sparse and imbalanced data to build robust models because there are only 20 countries out of 200 that have FTA in force with the US. By resampling the FTA in-force data, we made the training set balanced. To make sure the quality of the models, we separated the train data into train and validation sets to conduct 5-fold cross-validation. Also, for the hyper-parameter tuning, we used grid search. By doing that, we reached R^2 0.85 for the best estimator. We pickled the model and loaded it into the app. In the app, by getting the country and FTA year as input, we fetch the data to build the test set from the neo4j database. Then the app enters the test set into the unpickled model to get the precision.