Graph Visualisation for Interaction between DGC for Different Medicinal Types

Mukul Vyas, Pushkar Maurya, Rahul Kishore Gorai
mukul23128@iiitd.ac.in, pushkar23069@iiitd.ac.in, rahul23070@iiitd.ac.in

Abstract

This project aims to analyze and visualize the interactions between diseases, genes, and chemicals within Ayurvedic and Chinese medicine. We apply natural language processing (NLP) techniques to clean and preprocess the data. Then, we replace disease names with symptoms to ensure data consistency. Next, we generate interaction datasets by mapping diseases to genes and genes to chemicals. Finally, we visualize these interactions using a Sankey diagram, providing a clear graphical representation of the connections between different elements in the dataset.

1 Introduction

The study of interactions between diseases, genes, and chemicals is crucial in understanding the mechanisms underlying various medical conditions and the effects of therapeutic interventions. Traditional medicinal systems, such as Ayurveda and Chinese medicine, offer valuable insights into these interactions through centuries of empirical observations and practices.

In this project, we focus on visualizing the interactions between diseases, genes, and chemicals within the frameworks of Ayurvedic and Chinese medicine. By leveraging natural language processing (NLP) techniques and data visualization tools, we aim to provide a comprehensive understanding of the complex relationships between these elements and their implications for traditional medical practices.

2 Methodology

2.1 Data Retrieval and Preprocessing

- **Mounting Google Drive**: The script begins by mounting Google Drive to the Colab environment. This allows access to files stored in Google Drive, enabling the use of data files directly from the drive.
- **Installing Required Package**: The script installs a specialized R package called `pubmed.mineR`. This package is designed for mining and analyzing data from PubMed, a biomedical literature database.
- **Loading Required Libraries**: Several R libraries are loaded to provide additional functionality:
  - `pubmed.mineR` is loaded to utilize its functions for PubMed data analysis.
  - `httr` is loaded to facilitate making HTTP requests, which can be useful for accessing web-based APIs.
  - `jsonlite` is loaded to handle JSON data, enabling the parsing and creation of JSON-formatted information.
- **Reading a CSV File**: The script reads a CSV file named `den.csv` into an R data frame. This file is expected to contain relevant PubMed data, such as PMIDs (PubMed IDs) and abstracts of research papers.

2.2 PubTator Result Generation

- **PubTator Result Generation**:
  - The script defines a function called `pubtator` which retrieves information from the PubTator API based on PubMed IDs (PMIDs).
  - It constructs a URL for accessing the PubTator API for each PMID provided.
  - It sends a GET request to the PubTator API and processes the response to extract relevant information such as genes, diseases, mutations, chemicals, and species mentioned in the corresponding PubMed article.
- **Processing PubTator Results**:
  - The script organizes the extracted information into a structured format, separating genes, diseases, mutations, chemicals, and species into distinct categories.
  - It removes duplicates from each category to ensure unique entries.
  - It constructs a data frame containing the PubMed ID (PMID) along with the extracted information for each category.
- **Iterating Through PMIDs**:
  - The script iterates through each PubMed ID (PMID) provided in the input CSV file.
  - For each PMID, it calls the `pubtator` function to retrieve and process information from the PubTator API.
  - It appends the processed results for each PMID to a cumulative data frame.
- **Saving Results to CSV**:
  - Once all PMIDs are processed, the script writes the accumulated results to a CSV file named `Pubtator_Result_Traditional_Medicine.csv`.
  - Each row in the CSV file corresponds to a PubMed article, with columns representing genes, diseases, mutations, chemicals, and species mentioned in the article.

2.3 Matching PMID with PMCID

1. The script uses pandas, a Python data manipulation library, to read the gene count CSV file and the main file downloaded from PubMed.
2. It merges the two data frames based on the common column 'PMID', which represents PubMed IDs.
3. For each gene, disease, and chemical, it aggregates the corresponding PMCID values into a list.
4. The resulting data frames are then written to new CSV files.

2.4 Preprocessing the New CSV File

1. The script specifies a file path for the new CSV file that was generated in the previous step.
2. It reads the CSV file into a DataFrame using pandas.
3. It checks for null values in any cell and drops the corresponding rows to ensure data quality.
4. The modified DataFrame is then written back to a new CSV file, overwriting the existing one.

2.5 Switch to R and Generate Sentences for Disease/Gene/Chemical

2.5.1 Generating Sentences for Disease Count

1. The script starts by reading the CSV file containing disease counts, extracting *DiseaseName* and *PMCID* columns into separate lists.
2. It defines a function `extract_numeric_part` to extract the numeric part from PMCID, considering that PMCID values may contain non-numeric characters.
3. PMCID values are processed using the defined function, and a list of lists is generated, storing numeric parts of PMCID values.
4. An empty data frame *results_df* is initialized to store the results.
5. The script iterates over each *DiseaseName* and corresponding PMCID list.
6. For each disease and PMCID, it calls the function `Give_Sentences_PMC` to generate sentences.
7. If sentences are generated, they are concatenated into a single string and added to the *results_df*.
8. Finally, the results data frame is written to a CSV file named `results_chinese_dh.csv`.

2.5.2 Generating Sentences for Gene Count

1. The script follows a similar process as for diseases, but this time it reads the CSV file containing gene counts and extracts *GeneName* and *PMCID* columns.
2. It iterates over each *GeneName* and corresponding PMCID list, generating sentences using the `Give_Sentences_PMC` function.
3. The results are stored in the *results_df* data frame and written to a CSV file named `results_Chinese_gn.csv`.

2.5.3 Generating Sentences for Chemical Count

1. This part of the script is analogous to the previous two sections but deals with chemical counts.
2. It reads the CSV file containing chemical counts, extracts *ChemicalName* and *PMCID* columns, and iterates over them to generate sentences.
3. The results are stored in the *results_df* data frame and written to a CSV file named `results_Chinese_ch.csv`.

2.6 Switch to Python and Perform Interaction Between Genes, Chemicals, and Diseases

2.6.1 Loading Dataframes

1. The script starts by loading three CSV files into Pandas DataFrames: one containing information about chemicals, another for genes, and the third for diseases.

2.6.2 Interaction Between Gene and Chemical

1. It iterates through each row of the chemicals and genes dataframes.
2. For each pair of rows with the same PMID, it checks if the sentences match.
3. If a match is found, it extracts relevant information such as PMCID, sentence, gene, chemical, interaction type, and regulation using regular expressions.
4. It appends this information to respective lists.
5. Finally, it creates a new dataframe with the extracted data and saves it to a CSV file named `interaction_chemical_gene.csv`.

2.6.3 Interaction Between Gene and Disease

1. This section is similar to the previous one but focuses on interactions between genes and diseases.
2. It iterates through each row of the diseases and genes dataframes.
3. For each pair of rows with the same PMID, it checks if the sentences match.
4. If a match is found, it extracts relevant information such as PMCID, sentence, gene, disease, interaction type, and regulation using regular expressions.
5. It appends this information to respective lists.
6. Finally, it creates a new dataframe with the extracted data and saves it to a CSV file named `interaction_Gene_Disease.csv`.

2.6.4 Removing Duplicate Rows

1. After generating interaction dataframes, the script removes duplicate rows to ensure each interaction is unique.

2.7 Applying Natural Language Processing

Tools Used: NLTK

Steps Taken:

1. **Downloading NLTK Resources:** Downloaded necessary resources such as `punkt` for tokenization, `wordnet` for lemmatization, and `stopwords` for removing common English words that do not add significant meaning to the text.
2. **Initializing Lemmatizer and Stemmer:** Created instances of `WordNetLemmatizer` and `PorterStemmer` to process text data.
3. **Defining Text Processing Functions:**
   - Implemented `stem_text` function to stem words in a text, reducing them to their root form.
   - Implemented `lemmatize_text` function to lemmatize words in a text, converting them to their base form.
4. **Cleaning DataFrames:**
   - Converted text to lowercase to ensure uniformity.
   - Applied lemmatization and stemming to the text data in the dataframes.
   - Removed duplicate rows to maintain a clean dataset.

2.8 Replacing Disease Names with Symptoms

Goal: Replace disease names with symptoms in the datasets to ensure consistency and accuracy in data representation.

Steps Taken:

1. **Loading Disease-Symptom Dataset:** Loaded a dataset (`dataset.csv`) containing mappings between diseases and their corresponding symptoms.
2. **Replacing Disease Names:**
   - Implemented a function `replace_disease_names` that iterates over each row of the interaction dataframes.
   - For each disease in the interaction dataframe, checked if it exists in the disease-symptom dataset.
   - If a disease was found, matched symptoms from the interaction dataframe with the dataset.
   - If more than 4 symptoms matched, replaced the disease name with the one from the dataset.

2.9 Generating Interactions

Goal: Create a comprehensive dataset representing interactions between diseases, genes, and chemicals from both Ayurvedic and Chinese medicine perspectives.

Steps Taken:

1. **Loading Data from CSV Files:** Loaded interaction data from CSV files for Ayurvedic and Chinese gene-disease interactions, and chemical-gene interactions.
2. **Cleaning DataFrames:** Applied the previously defined cleaning functions to the loaded datasets to ensure text uniformity and remove duplicates.
3. **Creating Interactions:**
   - Generated interactions from Ayurvedic and Chinese medicine groups to diseases.
   - Generated interactions from diseases to genes.
   - Generated interactions from genes to chemicals.
4. **Ensuring Comprehensive Gene Associations:**
   - Identified genes without disease or chemical associations.
   - Added placeholders for genes without associated diseases or chemicals to ensure all genes were represented in the interaction dataset.
5. **Compiling Interactions:** Compiled all generated interactions into a single DataFrame.
6. **Saving the Interaction Dataset:** Saved the interaction dataset to a CSV file (`interactions.csv`).

2.10 Generating Sankey Diagram from Interactions

Goal: Visualize the interactions between Ayurvedic/Chinese medicine, diseases, genes, and chemicals using a Sankey diagram.

Steps Taken:

1. **Loading Interaction Data:** Loaded the compiled interaction data from the `interactions.csv` file.
2. **Grouping Data:** Grouped the interaction data by 'Source' and 'Target' columns, summing the 'Value' to aggregate interactions.
3. **Converting Data into Nodes and Links:**
   - Created lists for labels, sources, targets, and values to structure the data for visualization.
   - Assigned unique indices to each label (source and target) to create a lookup for the Sankey diagram.
4. **Creating Sankey Diagram:** Used Plotly to create the Sankey diagram, specifying node colors, orientations, and domain.
5. **Updating Chart Layout:** Adjusted the layout of the chart, including title, font size, font color, and height for better readability.
6. **Saving the Graph:** Saved the resulting Sankey diagram as an HTML file (`scatter_plot.html` for the subset and `scatter_plot_lg.html` for the whole data).
7. **Displaying the Graph:** Displayed the graph for visual verification.

3 Result

The analysis revealed significant interactions between diseases, genes, and chemicals within Ayurvedic and Chinese medicinal frameworks. The resulting Sankey diagram visually represents these interactions, highlighting the intricate relationships and pathways that connect various elements in the dataset. This visualization provides a clear and intuitive understanding of the complex connections inherent in traditional medical practices.

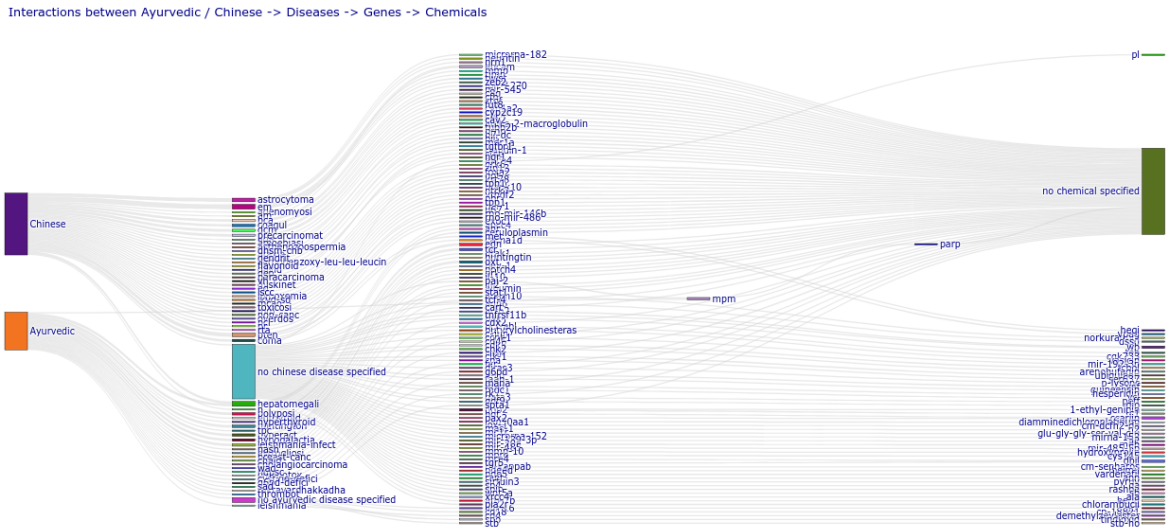

Figure 1: Sankey Diagram Visualizing Interactions Between Diseases, Genes, and Chemicals for Ayurveda and Traditional Chinese Medicines

4 Conclusion

In this project, we successfully analyzed and visualized the interactions between diseases, genes, and chemicals within the frameworks of Ayurvedic and Chinese medicine. By leveraging natural language processing (NLP) techniques and data visualization tools, we created a comprehensive dataset and a visual representation of these complex relationships.

Initially, we retrieved and preprocessed data from PubMed using specialized packages and APIs. This allowed us to extract relevant information about genes, diseases, and chemicals mentioned in biomedical literature. We then matched PMIDs with PMCIDs to ensure data consistency and completeness.

Our next step involved generating sentences for disease, gene, and chemical counts using R. This provided a structured way to understand the context in which these entities were mentioned in the literature. We then switched to Python to explore interactions between genes, chemicals, and diseases, creating interaction datasets and removing duplicates to maintain data integrity.

To enhance the quality of our dataset, we applied NLP techniques using the NLTK library. This included tokenization, lemmatization, and stemming to standardize the text data. Additionally, we replaced disease names with corresponding symptoms to ensure consistency and accuracy in data representation.

We then generated a comprehensive interaction dataset by mapping diseases to genes and genes to chemicals, ensuring all relevant associations were captured. This dataset was used to create a Sankey diagram, which visually represents the interactions between Ayurvedic/Chinese medicine, diseases, genes, and chemicals.

The Sankey diagram provided an intuitive way to understand the flow and connections between different elements in the dataset, highlighting the complex relationships inherent in traditional medical practices. This visualization can serve as a valuable tool for researchers and practitioners to explore and understand the interactions within these medicinal systems.

Overall, this project demonstrates the power of combining NLP techniques with data visualization to analyze and present complex biomedical data. The methods and tools used here can be applied to other domains and datasets, offering insights into various fields of research and practice.