**COMP 262**

**NATURAL LANGUAGE PROCESSING & RECOMMENDER SYSTEMS**

LAB 03

DINH HOANG VIET PHUONG – 301123263

**EXERCISE 1**

Import Necessary Libraries:

* import json: Imports the JSON library for parsing JSON data, enabling conversion between JSON strings and Python objects.
* import pandas as pd: Imports the Pandas library as pd, a powerful tool for data analysis and manipulation, particularly suited for structured data operations.
* from apyori import apriori: Imports the apriori function from the apyori library, which implements the Apriori algorithm for mining frequent itemsets and association rules from transaction data.
* from collections import Counter: Imports the Counter class from Python's standard collections module, designed for counting occurrences of hash able objects in an efficient way.

A black rectangular object with white border

Description automatically generated

Load the data

A black rectangular object with a black border

Description automatically generated

This code segment performs initial data exploration on a recipes dataset:

* DataFrame Conversion: Converts the raw data into a Pandas DataFrame named recipes, enabling efficient data manipulation.
* Counting Instances and Cuisines: Calculates the total number of recipes (39,774) and identifies 20 unique cuisines within the dataset.
* Cuisine Distribution Table: Creates a table listing each cuisine type alongside its corresponding recipe count. Italian cuisine tops the list with 7,838 recipes, while Brazilian cuisine has the least with 467 recipes.
* Output: Displays the total recipe count, number of cuisines, and a detailed table showcasing the distribution of recipes across different cuisines, providing insights into the dataset's diversity and composition.

A screenshot of a computer program

Description automatically generated

Average Ingredients Analysis SummaryThis code snippet calculates the average number of ingredients used in recipes across different cuisines and presents the findings in a structured format:

* Ingredient Count Addition: Adds a new column ingredient\_count to the recipes DataFrame, which represents the count of ingredients in each recipe. This is achieved by applying the Len function to each entry in the ingredient’s column.
* Calculating Averages: Groups the data by cuisine and computes the mean number of ingredients for each cuisine. The result is a DataFrame with cuisines and their corresponding average ingredient counts.
* Renaming Columns: The resulting DataFrame's columns are renamed to 'Cuisine Type' and 'Average Number of Ingredients' for clarity.
* Output Display: Prints a table listing each cuisine alongside its average number of ingredients. The table reveals that Cajun Creole cuisine uses the most ingredients on average (approximately 12.62), whereas Irish cuisine uses the fewest (around 9.30).

A screenshot of a computer

Description automatically generated

This code segment identifies the most frequently used ingredients across all recipes in the dataset and displays the top 10:

* Flattening Ingredient Lists: Compiles a single list of all ingredients used across all recipes by flattening the nested lists in the recipes['ingredients'] column.
* Counting Ingredient Occurrences: Utilizes the Counter class from the collections module to count how many times each ingredient appears in the dataset.
* Identifying Most Common Ingredients: Transforms the count data into a Pandas DataFrame named most\_common\_ingredients, with two columns: 'Ingredient' and 'Frequency'. This DataFrame is sorted by frequency, showcasing the most commonly used ingredients at the top.
* Output Display: Prints the top 10 most common ingredients, with 'salt' leading the list at 18,049 occurrences, followed by 'onions' and 'olive oil', each with 7,972 occurrences.

A screen shot of a computer

Description automatically generated

This script features a function designed to uncover the most common ingredients within recipes of a specified cuisine, showcasing these ingredients and their frequencies. Following this, it iteratively applies the function across all unique cuisines in the dataset to highlight culinary preferences and staple ingredients unique to each cuisine:

* Function Definition most\_common\_ingredients\_by\_cuisine: This function accepts a cuisine type and an optional parameter n (defaulting to 5) to specify the number of top ingredients to display. It filters the recipes dataset for the given cuisine, aggregates all ingredients from these recipes, and then counts the occurrences of each ingredient. The n most common ingredients and their counts are formatted into a DataFrame and printed.
* Aggregating Cuisine-Specific Ingredients: For each specified cuisine, it compiles a list of ingredients used across all its recipes. This is achieved by iterating over the 'ingredients' column of the filtered DataFrame, which contains only recipes belonging to the specified cuisine.
* Counting and Sorting Ingredients: Uses the Counter object to count the frequency of each ingredient within the cuisine-specific list, identifying the most common ingredients. The results are sorted by frequency in descending order.
* Iterative Cuisine Analysis: The script iterates over each unique cuisine type present in the recipes DataFrame. For each cuisine, it calls the most\_common\_ingredients\_by\_cuisine function, passing the cuisine name and the desired number of top ingredients to display (in this case, 10).
* Outputs: For each cuisine, the function prints a list of the top n ingredients most frequently used in that cuisine's recipes, providing insights into the foundational ingredients that characterize each cuisine's flavor profile and cooking practices.

A screen shot of a computer

Description automatically generated

This function performs Apriori analysis on recipes of a specific cuisine within a given dataset, aiming to discover common ingredient combinations and association rules that meet specified criteria. Here's a breakdown of how it works:

Function: apriori\_analysis

Parameters:

* cuisine: A string specifying the cuisine to analyze.
* recipes: A DataFrame containing recipe data.

Process:

1. Filtering Cuisine-Specific Recipes: The function first isolates recipes of the specified cuisine by matching the cuisine column with the provided cuisine parameter, ensuring case-insensitive comparison. If no recipes match the cuisine, the function returns None for both outputs, indicating no analysis was possible.

2. Preparation for Apriori:

* Extracts lists of ingredients (transactions) from the filtered recipes.
* Determines a dynamic support value based on the number of recipes to ensure the Apriori analysis is adapted to the dataset size.

3. Apriori Analysis Execution: Runs the Apriori algorithm with parameters for minimum support, confidence, lift, and length to find frequent ingredient combinations and derive association rules from them.

4. Extracting and Formatting Results:

* Identifies the top 2 ingredient sets by support values, formatting them for display.
* Extracts rules with a lift greater than 2, formatting these rules to highlight the base ingredients, added ingredients, confidence, and lift.

Outputs:

* top\_sets\_formatted: A list of strings representing the top ingredient combinations found in the specified cuisine's recipes, along with their support values.
* rules\_formatted: A list of strings describing the association rules with a lift greater than 2, including the base and added ingredients, confidence, and lift values.

This function is useful for understanding common ingredient pairings and potential recipe suggestions within a particular cuisine, providing insights into culinary practices and preferences.

A screen shot of a computer program

Description automatically generated

This code snippet outlines a user-interactive loop designed for querying Apriori analysis results on different cuisines based on a dataset contained in the recipes DataFrame. It facilitates user engagement by prompting for a cuisine type, processing the request, and displaying relevant Apriori analysis outcomes, such as common ingredient combinations and association rules with significant lift values:

* Cuisine Input: The loop starts by asking the user to enter a cuisine type. The input is formatted to be case-insensitive by converting it to lowercase. If the user enters 'bye', the loop ends with a goodbye message.
* Input Validation: Checks if the entered cuisine is present in the recipes DataFrame. If not, it informs the user that there are no recommendations for the entered cuisine and prompts for another input.
* Apriori Analysis: For valid cuisine inputs, it calls the apriori\_analysis function, passing the user's cuisine choice and the recipes DataFrame. The function returns two lists:
* top\_ingredients\_sets: Contains formatted strings of the most common ingredient combinations for the specified cuisine.
* rules\_with\_lift\_over\_2: Contains formatted strings of association rules where the lift value is greater than 2.
* Results Handling: If the analysis returns results, it prints the top ingredient combinations and rules with a lift greater than 2, providing insights into popular ingredients and potentially interesting ingredient pairings within the specified cuisine.
* Iteration: After displaying the results, the loop prompts the user for another cuisine type, allowing for continuous exploration until the user decides to exit by typing 'bye'.

This interactive loop is particularly useful for users interested in culinary data analysis, offering a direct and engaging way to explore common ingredient combinations and significant association rules across various cuisines.

A screenshot of a computer program

Description automatically generated

NOTE: Because the amount of data is too large (39774 instances), I reduced the capacity of the data to 124 instances to use for the final step (user interface).

**EXERCISE 2**

Importing pandas: Initiates the process by importing the pandas library, essential for data manipulation.

1. **Load the dataset**

Loading the Dataset: Utilizes pd.read\_json to load a compressed JSON Lines file (meta\_Digital\_Music.json.gz) into a DataFrame, facilitating the handling of large datasets.

Previewing Data: Employs the .head() method on the DataFrame to display the first few rows, providing a snapshot of the data's structure and content.

A screen shot of a computer

Description automatically generated

The DataFrame includes various fields such as:

* category, tech1, description, and fit: These columns seem to be empty or hold placeholder data.
* title and brand: Contains the title of the music album and the associated brand or artist's name.
* also\_buy and also\_view: Suggest other items customers also bought or viewed, likely meant for recommendation purposes.
* rank: Reflects the album's sales rank in the CDs & Vinyl category.

A screenshot of a computer screen

Description automatically generated

* price: Lists the price of the music item.
* asin: Amazon Standard Identification Number, a unique block of letters and/or numbers that identifies items.
* imageURL and imageURLHighRes: Contains URLs to album cover images, with a high-resolution option.
* date and details: These columns are either not fully visible or contain 'NaN', indicating missing data.

This DataFrame serves as the initial dataset for potential analysis, cleaning, or further manipulation. The presence of 'NaN' suggests there might be incomplete data points that require attention. The included image URLs point to external resources, likely hosted on Amazon, given the 'ssl-images-amazon' URL structure. The price column contains numerical values formatted as strings, prefixed with a dollar sign, indicating the listed price of albums in USD.

A screenshot of a computer

Description automatically generated

1. **Data Exploration**

This is a block of Python code designed to create an exploration summary of a dataset named songs\_phuong:

* Initialization: A dictionary named exploration\_summary\_corrected is defined with empty lists for columns: Column, Data Type, Null Count, Empty String Count, Empty List Count, and Unique Values or Note. This dictionary is structured to record data characteristics for each column in the dataset.
* Data Examination Loop: A for-loop iterates over the columns of the songs\_phuong DataFrame. For each column, the code performs the following:
* Appends the column's name to the Column list.
* Determines the data type of the column, appending it to the Data Type list.
* Counts the number of null values, appending the count to the Null Count list.
* If the column is of object-type (typically strings or lists in pandas), it counts the number of empty strings and empty lists, appending these counts to Empty String Count and Empty List Count respectively.
* Unique Value Calculation: The code attempts to calculate the number of unique values in each column. If a column contains unshushable types, which cannot be counted using the standard method, it records a note instead of the count.
* Summary DataFrame: The dictionary is then converted into a pandas DataFrame called exploration\_df\_corrected, which is formatted for easy viewing. This DataFrame summarily describes the dataset with a comprehensive overview of each column's characteristics.
* Output: Finally, the exploration DataFrame is displayed, providing a tabular summary of the dataset's structure, which includes data types, counts of null and empty values, and the uniqueness of the data in each column.

**A screen shot of a computer program

Description automatically generated**

The provided image displays a DataFrame exploration\_df\_corrected which summarizes the characteristics of the songs\_phuong dataset, following the execution of the described code block:

* The DataFrame consists of 19 rows, each corresponding to a different column from the songs\_phuong dataset, and 6 columns providing details on data types and value counts.
* Data Type: All columns are of object type, indicating they contain data which could be strings, lists, or other objects.
* Null Count: No columns have null values, suggesting that while there may be empty strings or lists, there are no completely missing entries.
* Empty String Count: Several columns contain a high number of empty strings, particularly tech1, fit, tech2, and date, with exactly 74,347 empty strings each, which may require cleaning.
* Empty List Count: The category, description, also\_buy, feature, also\_view, imageURL, imageURLHighRes, and details columns contain numerous entries that are empty lists, indicating potential placeholders or data that was not collected.
* Unique Values or Note: For columns containing hashable types, the unique value counts are provided. Some columns contain unhashable types, so a unique count was not possible, noted in the summary.

This DataFrame effectively captures the preliminary data quality and structure, essential for informing further data cleaning and analysis steps. The presence of empty strings and lists is noteworthy and may necessitate attention before in-depth analysis.

**A screenshot of a computer

Description automatically generated**

It describes the steps for refining a dataset by removing irrelevant columns and filtering out rows with missing critical information:

* Dropping Columns: The columns deemed irrelevant or containing many empty entries (tech1, tech2, fit, similar\_item, date, imageURLHighRes) are identified to be dropped. This is achieved using the drop method on the original DataFrame songs\_phuong, creating a new DataFrame songs\_phuong\_filtered.
* Removing Rows with Missing Information: It then filters the DataFrame to remove rows where the title, brand, or asin fields are empty. This step is crucial as these columns likely contain essential information for the dataset's purpose, and their absence could affect the integrity of any analysis performed.
* Previewing the Data: Finally, the .head() method is called on the filtered DataFrame to display the first few rows, allowing for a quick inspection to ensure the data has been cleaned as intended.

These preprocessing steps are standard data cleaning operations that enhance the dataset's quality, preparing it for more reliable and accurate analysis.

**A screen shot of a computer program

Description automatically generated**

The image shows the first few rows of the songs\_phuong\_filtered DataFrame after applying the data cleaning steps specified in the provided code. The visible columns are: category, description, title, also\_buy, brand, feature, rank, also\_view, main\_cat, price, asin, imageURL, details.

This DataFrame no longer includes the columns tech1, tech2, fit, similar\_item, date, and imageURLHighRes, as they were identified for removal. It seems to display music albums with related purchasing options (also\_buy, also\_view), along with metadata like brand, price, and sales rank.

All rows where the title, brand, or asin was an empty string have also been removed, ensuring that the remaining dataset includes only records with critical information intact. This cleaned dataset is now more streamlined for analysis, with reduced noise and irrelevant data. The imageURL column still exists, indicating it contains relevant information such as links to album cover images.

The displayed DataFrame is the result of a necessary preprocessing step to enhance data quality, which is especially important for accurate analysis, modeling, or recommendation system development.

**A screenshot of a computer program

Description automatically generated**

When building a recommender system for digital music metadata, it's essential to choose columns that can contribute to meaningful recommendations. Based on the dataset, here are the columns I would consider and the reasons why:

* title: It's the core attribute of any music item, indicating what the album or song is. Titles can be used for content-based filtering, matching similar items by name or associated metadata.
* brand: This typically refers to the artist or label. It's crucial for recommendations because customers often have preferences for certain artists, and this can be used for collaborative filtering by connecting users with similar artist preferences.
* also\_buy: These are items that customers also purchased alongside the given item. This column is valuable for collaborative filtering, as it directly indicates user preferences and purchasing patterns.
* also\_view: This shows items that customers also viewed. While not as strong an indicator of purchasing intent as 'also\_buy', it's still a useful feature for a recommender system, potentially for an item-item collaborative approach.
* feature: If this column contains information about the music or album features, it could be used for content-based recommendations to find items with similar characteristics.
* rank: The sales rank indicates popularity, which can be an excellent predictor for recommending items to users who prefer popular choices.
* price: Price can be a filtering criterion since users may have specific budgets or price sensitivities.
* main\_cat: The main category of the item is useful to ensure recommendations are within the same broader category, like genre in this case.
* asin: The Amazon Standard Identification Number is unique to each item and is necessary for uniquely identifying and linking different items within the recommendation system.

The imageURL could be used to display the album cover to the user in the recommender system interface, enhancing user experience, but it wouldn't directly influence the recommendation algorithm.

Columns like description, category, and details might provide additional context for content-based filtering if they contain relevant metadata about the music (e.g., genre, sub-genre, mood, instruments, etc.). However, their use would depend on whether this information is indeed present and consistently formatted.

The selection of these columns is with the goal of creating a system that can provide recommendations based on user behavior (collaborative filtering) and item characteristics (content-based filtering). The key is to utilize both explicit data (like title and brand) and implicit feedback (also\_buy, also\_view) to generate personalized recommendations.

1. **Feature Engineering:**

This code block is focused on feature engineering for building a content-based recommender system using text data from the songs\_phuong\_filtered DataFrame. It outlines the following steps:

* Text Combination: It creates a new column combined\_text by concatenating the title and description columns, ensuring that missing descriptions are treated as empty strings. This enriches the dataset with combined textual information for better content analysis.
* TF-IDF Vectorization: The code uses TfidfVectorizer from scikit-learn to convert the combined text into a TF-IDF matrix, a numerical representation that emphasizes important words within each document while accounting for the frequency of these words across the entire dataset. English stop words are excluded to reduce noise in the data.
* Cosine Similarity Computation: Next, the code computes the cosine similarity between all TF-IDF vectors. Cosine similarity is a metric used to measure how similar the documents are irrespective of their size, in this case, how similar songs are based on their combined text content.
* Recommendation Function: The get\_recommendations function takes a song title and cosine similarity matrix as input. It finds the song's index in the DataFrame, retrieves the cosine similarity scores for that song against all others, sorts them to find the top scores, and returns the titles of the top 10 most similar songs.
* Example Usage and Saving Recommendations: It demonstrates how to use the function by getting recommendations for the first song title in the filtered dataset and then saving these recommendations to a CSV file at a specified location (D:\Download\song\_recommendations.csv).

The output recommendations will be a pandas Series object containing the titles of the top 10 recommended songs based on the text similarity to the first song in the dataset. This list is what would be presented to a user in a recommender system when they are viewing or listening to the first song.

The feature engineering steps performed in this code are crucial for setting up a robust content-based filtering system that can automatically recommend similar songs to users based on the textual content associated with each song in the dataset.

A computer screen shot of text

Description automatically generated

The image displays the output of the get\_recommendations function from the previously described code block. It shows a pandas Series with indices and the titles of the recommended songs. This Series is the result of applying the TF-IDF and cosine similarity-based recommendation process to the first song in the dataset.

These recommendations are the top 10 songs considered most textually similar to the first song, titled "Volume 1". The titles include various volumes, suggesting the original song might be part of a collection or series. There's also a title "Here I Am-Collection", which may be another collection or album series.

The indices are the positions of these songs within the original DataFrame, which can be used to retrieve additional information about these recommendations if needed. This output can be used to present a list of suggested songs to users in a recommender system interface.

A screenshot of a computer

Description automatically generated

1. **Song recommendation**

This code snippet defines an interactive recommender function named recommend\_songs\_interactively for suggesting songs to a user:

* The function enters an infinite loop, prompting the user to enter a song title for which they would like to get recommendations. The user can also type 'exit' to quit the interactive session.
* If the user chooses to exit, the loop is terminated with a farewell message.
* When a song title is entered, the function checks if the title exists within the songs\_phuong\_filtered DataFrame.
* If the song is found, it calls the get\_recommendations function with the user's input, retrieves the recommendations, and prints them out in an enumerated list.
* If the song is not found, it informs the user that there are no recommendations available for the input provided.
* The function is intended to be called to begin interacting with the user, allowing them to continuously input song titles and receive recommendations until they decide to stop by typing 'exit'.

When you run the function recommend\_songs\_interactively(), it will engage you in a dialogue asking for a song title and return song recommendations based on the dataset and the similarity scores computed previously. It’s designed to be user-friendly and requires no prior knowledge of the code or dataset to use.

A screen shot of a computer program

Description automatically generated

It shows a user's interaction with the system, including song title inputs for recommendation and the system's responses:

* The user enters the title "grehte," which is not found in the dataset, and the system responds with a message indicating no recommendations can be made.
* The user then requests recommendations for "So You Wanna Go Back to Egypt," and the system successfully provides a list of top recommendations, which include several variations of "I Wanna 1-2-1 with You" and other similarly titled songs, suggesting that the system is picking up on keywords like "wanna."
* Another request is made for "Early Works - Dallas Holm." The system provides recommendations with religious themes, such as "Saved, Saved, Saved" and "The Name of Jesus," which align with the input song's likely theme and content.
* Finally, the user types 'exit', and the system acknowledges the command by exiting the recommender system with a goodbye message.

A screenshot of a computer screen

Description automatically generated