IR ASSIGNMENT 3

This code snippet demonstrates a series of data preprocessing steps applied to a DataFrame read from a CSV file. Let's break down each part of the code and discuss its purpose in detail:

1. Importing Libraries

python

```
import pandas as pd
```

This line imports the pandas library under the alias pd. Pandas is a powerful library in Python used for data manipulation and analysis.

2. Reading CSV File

python

```
output_csv = "/Users/DELL/Desktop/ir /Assignment 3/final_metadata.csv"

df = pd.read_csv(output_csv)
```

- output_csv: Specifies the path to the CSV file containing the dataset.
- pd.read_csv(output_csv): Reads the CSV file into a pandas DataFrame named df.

3. Pre-processing Steps

Handling Missing Values:

python

```
df.dropna(subset=['title'], inplace=True)
```

- dropna(subset=['title']): Drops rows with missing values in the 'title' column.
- inplace=True: Modifies the DataFrame in place, meaning changes are applied directly to df.

Handling Duplicates:

python

```
df.drop_duplicates(subset=['title'], keep='first', inplace=True)
```

- drop_duplicates(subset=['title'], keep='first'): Drops duplicate rows based on the 'title' column while keeping the first occurrence.
- inplace=True: Modifies the DataFrame in place.

Other Pre-processing:

python

```
df['title'] = df['title'].str.lower()
```

- df['title'].str.lower(): Converts the 'title' column values to lowercase.
- df['title'] = ...: Assigns the lowercase values back to the 'title' column in the DataFrame.

4. Filtering Rows

python

```
headphones_df = df[df['title'].str.contains('headphone', case=False)]
```

- df['title'].str.contains('headphone', case=False): Returns a boolean Series indicating whether each 'title' contains the substring 'headphone' (case insensitive).
- df[...]: Filters the DataFrame rows where the condition is True and assigns the result to a new DataFrame headphones_df.

5. Counting Rows

python

```
num_headphones = len(headphones_df)

print("Number of rows containing 'headphones' in the 'Title' column after pre-processing:", num_headphones)
```

- len(headphones_df): Calculates the number of rows in the filtered DataFrame headphones_df.
- Prints the count of rows containing 'headphones' in the 'Title' column after pre-processing.

Summary

This code snippet reads a CSV file into a pandas DataFrame, performs data preprocessing steps such as handling missing values and duplicates, converts 'title' column values to lowercase, filters rows containing 'headphones' in the 'title' column, and finally counts the number of such rows. These preprocessing steps are common in data analysis tasks to ensure data quality and prepare the dataset for further analysis or modeling.

1. Introduction

Recommender systems have become integral to various online platforms, including e-commerce websites, streaming services, and social media platforms. These systems analyze user interactions and preferences to generate personalized recommendations, thereby enhancing user experience and engagement.
In this report, we focus on improving a recommender system using Non-Negative Matrix Factorization (NMF) and K-fold cross-validation techniques. We aim to provide a detailed analysis of the data preprocessing steps, normalization techniques, evaluation methodologies, and visualization of results.

---

2. Data Preprocessing and Normalization

Data preprocessing is a crucial step in building recommender systems, involving the cleaning, transformation, and normalization of raw data. In our case, we start by loading the dataset containing user-item interactions, typically represented as a DataFrame with columns such as reviewerID, asin, and overall rating.
The provided code snippets demonstrate how to preprocess the data by creating a user-item rating matrix using the pandas library. This matrix represents users' ratings for different items, where missing values are filled

with zeros. Additionally, we apply min-max scaling to normalize the ratings, ensuring that all ratings fall within a specified range (usually 0 to 1). Normalization is essential for ensuring that the ratings across different users and items are comparable and consistent. By scaling the ratings to a common range, we mitigate biases and variations in the data, leading to more accurate recommendations.

---

3. User-Item Rating Matrix Analysis

Once the data is preprocessed and normalized, we perform an analysis of the user-item rating matrix to gain insights into user preferences and item popularity. This analysis involves exploring various metrics and visualizations to understand the distribution of ratings, user engagement, and item popularity.

Using pandas and matplotlib libraries, we can generate descriptive statistics, histograms, and heatmaps to visualize the distribution of ratings and identify trends in user behavior. For example, we may observe that certain items receive consistently high ratings, while others have more varied ratings.

Understanding user-item interactions is crucial for building effective recommender systems. By analyzing the rating matrix, we can identify patterns, outliers, and potential biases that may impact the quality of recommendations.

---

4. Cosine Similarity Calculation

Cosine similarity is a popular technique used in recommender systems to measure the similarity between items based on their ratings. It quantifies the cosine of the angle between two vectors, representing the ratings of two items by users.

In our implementation, we compute cosine similarity between items to identify items that are closely related or similar in terms of user preferences. This involves calculating the cosine similarity matrix, where each entry represents the similarity score between two items.

The cosine similarity matrix serves as a valuable resource for generating recommendations by identifying items that are likely to be preferred by users who have interacted with similar items. By recommending similar items, we can enhance user satisfaction and increase engagement.

---

5. K-fold Cross-validation for Item-Item Recommender Systems

Evaluation is a critical aspect of building recommender systems, as it allows us to assess the accuracy and effectiveness of the recommendations. K-fold cross-validation is a popular technique used to evaluate the performance of machine learning models, including recommender systems.

In our approach, we split the user-item rating matrix into K folds and train the NMF model on K-1 folds while evaluating its performance on the remaining fold. This process is repeated K times, with each fold serving as both training and validation data.

The provided code snippets demonstrate how to implement K-fold cross-validation using the sklearn library. We compute the Mean Absolute Error (MAE) for each fold and average the results to obtain an overall measure of the model's performance.

---

6. Plotting MAE against K for Item-Item Recommender Systems

To visualize the performance of the recommender system, we plot the Mean Absolute Error (MAE) against the number of components (K) for item-item recommender systems. The resulting plot provides insights into the optimal value of K for minimizing prediction errors.

Using matplotlib, we create a line plot where the x-axis represents the number of components (K) and the y-axis represents the MAE. By analyzing the plot, we can identify the value of K that results in the lowest MAE, indicating the optimal complexity of the model.

Visualizing the MAE against K helps us understand the trade-off between model complexity and prediction accuracy. A higher value of K may lead to

overfitting, while a lower value may result in underfitting. Finding the right balance is essential for building a reliable recommender system.

---

7. Conclusion

In conclusion, this report provides a comprehensive overview of the steps involved in analyzing and improving recommender systems using NMF and K-fold cross-validation techniques. By following these methodologies, organizations can develop more accurate and reliable recommender systems to meet the diverse needs of their users.
By preprocessing the data, normalizing ratings, analyzing user-item interactions, computing cosine similarity, and evaluating the model's performance, we can build robust recommender systems that deliver personalized recommendations and enhance user satisfaction.