Zee Recommender Systems

Akanksha Trivedi

Table of Contents

Define Problem Statement and Formatting the Data

Problem Definition:

We are tasked with building a movie recommendation system using user ratings and movie metadata. We will format the data, clean it, and merge the datasets into one consolidated dataframe to perform further analysis.

```python
import pandas as pd

# Load the datasets
users = pd.read_csv('/content/zee-users.dat', sep='::', engine='python', header=None, names=['UserID', 'Gender', 'Age', 'Occupation', 'Zip-code'])
movies = pd.read_csv('/content/zee-movies.dat', encoding='ISO-8859-1', sep='::', engine='python', header=None, names=['Movie ID', 'Title', 'Genres'])
ratings = pd.read_csv('/content/zee-ratings.dat', sep='::', engine='python', header=None, names=['UserID', 'Movie ID', 'Rating', 'Timestamp'])

# Merge datasets
merged_df = pd.merge(pd.merge(ratings, users, on='UserID'), movies, on='Movie ID')
print(merged_df.head())
```

```
   UserID  Movie ID  Rating  Timestamp Gender  Age  Occupation Zip-code  \
0       1      1193       5  978300760      F    1          10    48067
1       1       661       3  978302109      F    1          10    48067
2       1       914       3  978301968      F    1          10    48067
3       1      3408       4  978300275      F    1          10    48067
4       1      2355       5  978824291      F    1          10    48067

                                    Title                        Genres
0  One Flew Over the Cuckoo's Nest (1975)                         Drama
1        James and the Giant Peach (1996)  Animation|Children's|Musical
2                     My Fair Lady (1964)               Musical|Romance
3                  Erin Brockovich (2000)                         Drama
4                    Bug's Life, A (1998)   Animation|Children's|Comedy
```

Double-click (or enter) to edit

Perform Exploratory Data Analysis (EDA), Data Cleaning, and Feature Engineering

Data Inspection and Cleaning:

We will check for any missing values, duplicates, and perform feature engineering (like extracting release year from the movie title).

```python
# Checking for missing values
merged_df.isnull().sum()
# Removing duplicates
merged_df.drop_duplicates(inplace=True)
# Extracting the release year from the movie title
merged_df['Release_Year'] = merged_df['Title'].str.extract(r'(\d{4})')
# Convert Release_Year to integer
merged_df['Release_Year'] = pd.to_numeric(merged_df['Release_Year'], errors='coerce')
#replace NaN values with the mean or median rating if needed
merged_df['Rating'] = pd.to_numeric(merged_df['Rating'], errors='coerce')
#replace NaN values with the mean or median rating if needed
merged_df['Rating'].fillna(merged_df['Rating'].mean(), inplace=True)
# Grouping by average rating and number of ratings
rating_stats = merged_df.groupby('Title').agg(avg_rating=('Rating', 'mean'), num_ratings=('Rating', 'count')).reset_index()
print(rating_stats.head())
```

```
                       Title  avg_rating  num_ratings
0        $1,000,000 Duck (1971)    3.666667            3
1          'Night Mother (1986)    3.571429            7
2     'Til There Was You (1997)    3.111111            9
3            'burbs, The (1989)    3.000000           42
4  ...And Justice for All (1979)    3.956522           23
```

Build a Recommender System Based on Pearson Correlation

Item-Based Recommender Using Pearson Correlation

We will create a pivot table of movie titles and user IDs, and then calculate the item similarity using Pearson Correlation.

```
# Create a pivot table of movies and users
pivot_table = merged_df.pivot_table(index='UserID', columns='Title', values='Rating')

# Calculate the Pearson Correlation for movie similarities
item_similarity = pivot_table.corr(method='pearson')

# Get recommendations based on Pearson correlation
movie_name = "Toy Story (1995)"
similar_movies = item_similarity[movie_name].sort_values(ascending=False).head(6)
similar_movies
```

	Toy Story (1995)
Title	
Mr. & Mrs. Smith (1941)	1.0
Children of the Corn III (1994)	1.0
Raw Deal (1948)	1.0
Bent (1997)	1.0
Slipper and the Rose, The (1976)	1.0
Blow-Out (La Grande Bouffe) (1973)	1.0

dtype: float64

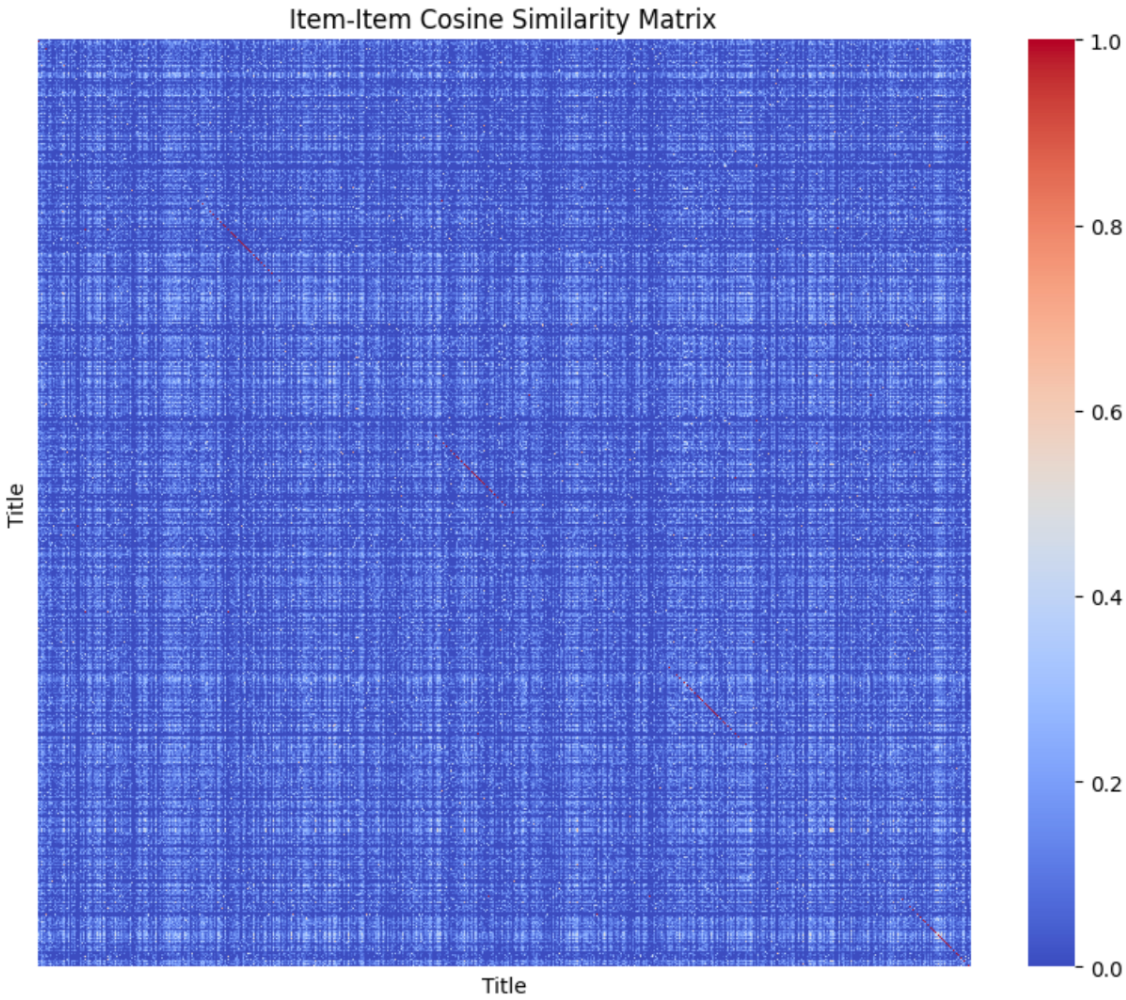
Build a Recommender System Based on Cosine Similarity

Cosine Similarity:

We will calculate the similarity between items (movies) and users using Cosine Similarity and visualize the user-item similarity matrix.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Heatmap for item-item similarity
plt.figure(figsize=(10, 8))
sns.heatmap(cosine_sim_df, cmap='coolwarm', xticklabels=False, yticklabels=False)
plt.title('Item-Item Cosine Similarity Matrix')
plt.show()
```



Build a Recommender System Based on Matrix Factorization

Matrix Factorization:

We will use the **Surprise** library to build a recommendation model using matrix factorization (SVD). We'll also evaluate the model using RMSE and MAPE.

```
!pip install scikit-surprise
from surprise import SVD, Dataset, Reader
from surprise.model_selection import train_test_split
from surprise import accuracy

# Prepare data for Surprise library
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(merged_df[['UserID', 'Title', 'Rating']], reader)

# Train-test split
trainset, testset = train_test_split(data, test_size=0.2)

# Build the SVD model
model = SVD()
model.fit(trainset)

# Make predictions
predictions = model.test(testset)

# Evaluate performance using RMSE and MAPE
rmse = accuracy.rmse(predictions)
print(f"RMSE: {rmse}")

# Calculate MAPE
mape = np.mean(np.abs((np.array([pred.est for pred in predictions]) - np.array([pred.r_ui for pred in predictions])) / np.array([pred.r_ui for pred in predictions]))) * 100
print(f"MAPE: {mape}")
```

```
Collecting scikit-surprise
  Downloading scikit_surprise-1.1.4.tar.gz (154 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.4/154.4 kB 3.0 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-surprise) (1.4.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-packages (from scikit-surprise) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-surprise) (1.13.1)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (pyproject.toml) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.4-cp311-cp311-linux_x86_64.whl size=2505174 sha256=0615c040bd933aba97efe38159b4510d5a987e91681ee887b9ccb7d7ea054012
  Stored in directory: /root/.cache/pip/wheels/2a/8f/6e/7e2899163e2d85d8266daab4aa1cdabec7a6c56f83c015b5af
Successfully built scikit-surprise
Installing collected packages: scikit-surprise
Successfully installed scikit-surprise-1.1.4
RMSE: 0.9181
RMSE: 0.9180541540076876
MAPE: 28.487686781979882
```

RMSE: 0.9181 **MAPE:** 28.49

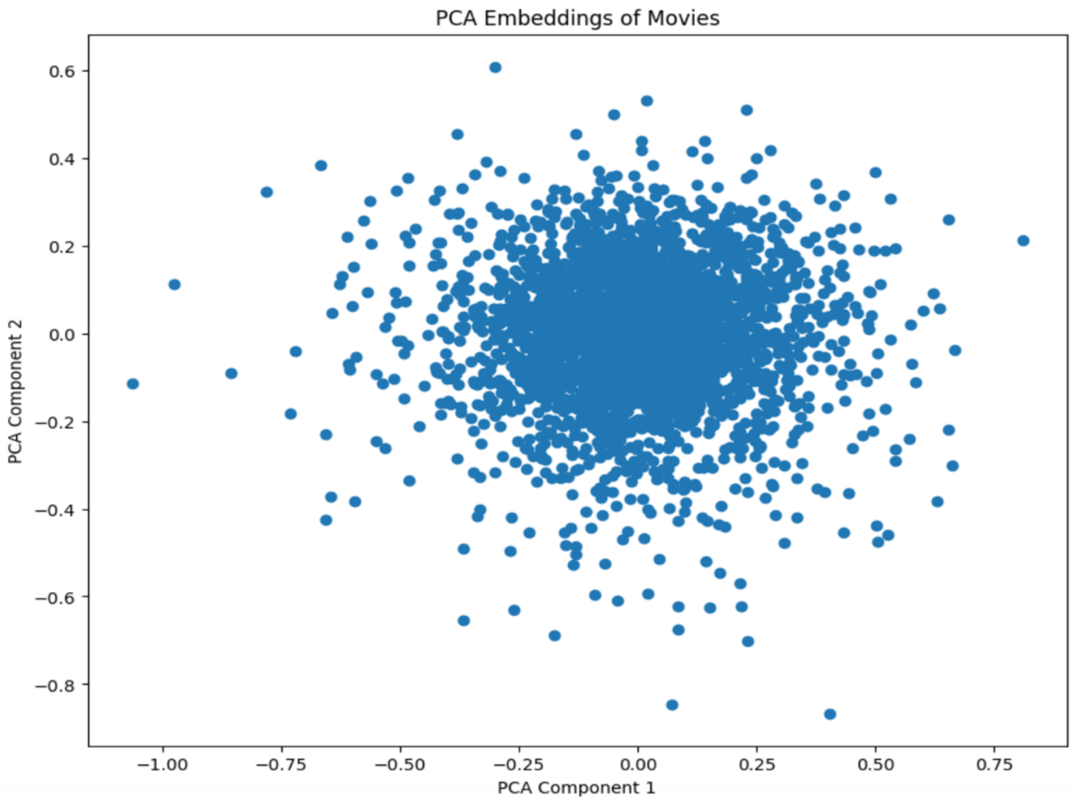
Embeddings and Visualization:

```python
# Getting embeddings from the SVD model for item-item similarities
item_embeddings = model.qi

# Visualizing the embeddings using PCA
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
reduced_embeddings = pca.fit_transform(item_embeddings)

# Plot the reduced embeddings
plt.figure(figsize=(10, 8))
plt.scatter(reduced_embeddings[:, 0], reduced_embeddings[:, 1])
plt.title("PCA Embeddings of Movies")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.show()
```



Pearson Correlation: -1 to +1

Cosine Similarity: 0 to 1

User-Based Approach

User-Based Recommender Using Pearson Correlation:

We will calculate the Pearson Correlation between users and recommend movies based on similar users.

```
# Create a pivot table where the index is UserID and columns are MovieID
pivot_table = merged_df.pivot_table(index='UserID', columns='Movie ID', values='Rating')

# Create user similarity matrix
user_similarity = pivot_table.corr(method='pearson')

similar_users = user_similarity['1'].sort_values(ascending=False).head(6)

print(similar_users)
```

```
Movie ID
1       1.0
2477    1.0
3731    1.0
3734    1.0
2704    1.0
2678    1.0
Name: 1, dtype: float64
```

Questionaries

Users of which age group have watched and rated the most number of movies?

```
merged_df.groupby('Age')['Rating'].count().sort_values(ascending=False)
```

Age	Rating
25	50790
18	32458
35	24102
45	11234
50	7743
56	3963
1	2678

Users belonging to which profession have watched and rated the most movies?

```
merged_df.groupby('Occupation')['Rating'].count().sort_values(ascending=False)
```

Occupation	Rating
4	131032
0	130499
7	105425
1	85351
17	72816
20	60397
12	57214

Most of the users in our dataset who've rated the movies are Male. (T/F)

```
merged_df['Gender'].value_counts()
```

Gender	count
M	753769
F	246440

dtype: int64

Most of the movies present in our dataset were released in which decade?

```
merged_df['Release_Year'] = pd.to_numeric(merged_df['Release_Year'])
(merged_df['Release_Year'] // 10 * 10).value_counts().idxmax()
```

```
1990
```

The movie with the maximum number of ratings is

0s

```python
rating_stats.loc[rating_stats['num_ratings'].idxmax(), 'Title']
```

```
'American Beauty (1999)'
```

Name the top 3 movies similar to ‘Liar Liar’ on the item-based approach. The Pearson Correlation

```python
similar_movies = item_similarity['Liar Liar (1997)'].sort_values(ascending=False).head(5).index[1:5].tolist()
print(similar_movies)
```

```
['Liar Liar (1997)', "Those Who Love Me Can Take the Train (Ceux qui m'aiment prendront le train) (1998)", 'Voyage of the Damned (1976)', 'Sacco and Vanzetti (Sacco e Vanzetti) (1971)']
```

Give the sparse ‘row’ matrix representation for the following dense matrix

[[1 0] [3 7]]

Sparse representation

(0, 0) 1 (1, 0) 3 (1, 1) 7