# Topic: Recommendation Engine

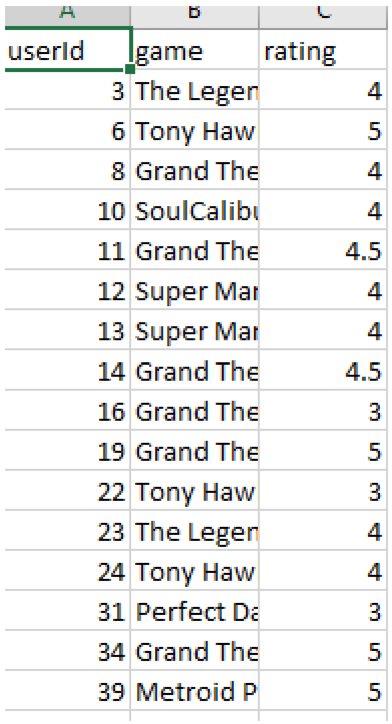
**Name:** ULLI VENKATA SAI KUMAR **Batch ID:**04072024HYD10AM

**Topic: Recommender Engine**

## Problem Statement: -

Q1) Build a recommender system with the given data using UBCF.

This dataset is related to the video gaming industry and a survey was conducted to build a recommendation engine so that the store can improve the sales of its gaming DVDs. A snapshot of the dataset is given below. Build a Recommendation Engine and suggest top-selling DVDs to the store customers.



| userId | game | rating |
|---|---|---|
| 3 | The Legen | 4 |
| 6 | Tony Haw | 5 |
| 8 | Grand The | 4 |
| 10 | SoulCalibu | 4 |
| 11 | Grand The | 4.5 |
| 12 | Super Mar | 4 |
| 13 | Super Mar | 4 |
| 14 | Grand The | 4.5 |
| 16 | Grand The | 3 |
| 19 | Grand The | 5 |
| 22 | Tony Haw | 3 |
| 23 | The Legen | 4 |
| 24 | Tony Haw | 4 |
| 31 | Perfect Da | 3 |
| 34 | Grand The | 5 |
| 39 | Metroid P | 5 |

```
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from scipy.sparse import csr_matrix
# Load the dataset to check its structure and understand its contents
game_data = pd.read_csv('game.csv')
# Display the first few rows of the dataset
game_data.head()
# Pivot the dataset to create a user-item matrix
user_game_matrix = game_data.pivot_table(index='userId', columns='game',
values='rating').fillna(0)
# Create a sparse matrix for the user-item matrix
user_game_sparse = csr_matrix(user_game_matrix)
# Compute cosine similarity between users
user_similarity = cosine_similarity(user_game_sparse)
# Create a DataFrame to store user similarities
user_similarity_df = pd.DataFrame(user_similarity, index=user_game_matrix.index,
columns=user_game_matrix.index)
# Function to recommend top games for a given user
def recommend_games(user_id, num_recommendations=5):
    # Get the similarity scores for the input user with all other users
    similar_users = user_similarity_df[user_id].sort_values(ascending=False)
    # Exclude the input user from the similar users list
    similar_users = similar_users.drop(user_id)
    # Find games rated by similar users that the input user has not rated
    similar_users_games = user_game_matrix.loc[similar_users.index]
    # Sum the ratings of the similar users
    weighted_ratings = similar_users_games.mul(similar_users,
axis=0).sum().sort_values(ascending=False)
    # Get games that the input user hasn't rated yet
    user_rated_games = user_game_matrix.loc[user_id]
    unrated_games = weighted_ratings[user_rated_games == 0]
    # Return the top recommendations
    return unrated_games.head(num_recommendations)
# Recommend top 5 games for user with userId 3 as an example
recommend_games(3, num_recommendations=5)
# Adjust recommendation function to focus on games with higher average ratings from similar
users
def recommend_top_games(user_id, num_recommendations=5, min_rating_threshold=3):
    # Get the similarity scores for the input user with all other users
    similar_users = user_similarity_df[user_id].sort_values(ascending=False)
    # Exclude the input user from the similar users list
    similar_users = similar_users.drop(user_id)
    # Find games rated by similar users that the input user has not rated
    similar_users_games = user_game_matrix.loc[similar_users.index]
    # Sum the ratings of the similar users
    weighted_ratings = similar_users_games.mul(similar_users,
axis=0).sum().sort_values(ascending=False)
    # Get games that the input user hasn't rated yet
    user_rated_games = user_game_matrix.loc[user_id]
    unrated_games = weighted_ratings[user_rated_games == 0]
    # Filter games with average ratings higher than the threshold
    top_recommendations = unrated_games[unrated_games >= min_rating_threshold]
    # Return the top recommendations
    return top_recommendations.head(num_recommendations)
# Recommend top 5 games for user with userId 3, focusing on higher-rated games
recommend_top_games(3, num_recommendations=5, min_rating_threshold=3)
```

## **Questions to Trigger Your thoughts:**

Q1. What are Recommendation Systems?
Recommendation systems are algorithms or models designed to suggest products, content, or services to users based on various factors like their past behaviors, preferences, or similar users' behaviors. These systems are widely used in platforms such as Netflix, Amazon, and YouTube to recommend movies, products, or videos, enhancing user engagement by delivering personalized experiences.

Q2. How are Knowledge-based Recommender Systems different from Collaborative and Content-based Recommender Systems?
Knowledge-based recommender systems use explicit knowledge about item attributes, user preferences, and constraints to make suggestions, often seen in domains like travel planning or complex services. In contrast:

- **Collaborative filtering** relies on patterns from user interactions (ratings, clicks) to suggest items, focusing on the behavior of similar users.
- **Content-based filtering** uses item features (such as genre, keywords) to recommend items based on a user's past preferences.

Q3. What is the difference between Collaborative *and* Content-based Recommender Systems*?*

- **Collaborative filtering**: Recommends items based on user behavior and preferences of similar users (e.g., "Users who liked this also liked that"). It doesn't require knowledge of the item's content but relies on user interactions.

- **Content-based filtering**: Recommends items based on their attributes and the preferences a user has shown for similar items (e.g., recommending books in a similar genre).

Q4. How does the surprise library work in the recommendation engine?
The surprise library is a Python library used for building recommendation systems. It supports various collaborative filtering algorithms (e.g., KNN, SVD) and is used for:

1. **Data loading and splitting** into training and testing sets.
2. **Model building**: Implementing algorithms like User-Based and Item-Based Collaborative Filtering.
3. **Evaluating performance**: Using metrics like RMSE, MAE.
4. **Predictions**: It helps generate recommendations for new users based on the trained model.

Q5. What are the three main types of recommendation engines?

- **Collaborative Filtering**: Based on user-item interactions (either user-based or item-based).

- **Content-Based Filtering**: Recommends items based on their features and the user's history.

- **Hybrid Systems**: Combines collaborative filtering, content-based, and/or knowledge-based methods to generate more accurate recommendations.

Q6. What is the logic behind the recommendation engine?
The logic behind a recommendation engine is to identify patterns in user behavior or item characteristics, then use those patterns to predict what a user would like. Algorithms work by analyzing either:

- **User preferences (Collaborative Filtering)**: Assumes users who agreed in the past will agree in the future.
- **Item similarity (Content-Based Filtering)**: Suggests items similar to what the user has interacted with.
- **Mixed approaches (Hybrid)**: Combines both approaches to reduce shortcomings of individual models.

Q7. What are the benefits of recommendation engines?

- **Personalization**: Offers a tailored experience to users, enhancing satisfaction and engagement.
- **Increased sales and retention**: Suggests relevant products/content, increasing conversion rates and retaining customers.
- **Efficient content discovery**: Helps users find new products or content they might not have discovered otherwise.
- **Data-driven decisions**: Provides insights into user behavior, preferences, and trends.

Q8. What is NLP usage in recommendation engines?
Natural Language Processing (NLP) is used in recommendation engines to:

- **Extract features**: From textual data such as reviews, item descriptions, or tags.
- **Sentiment analysis**: To understand user preferences based on feedback.
- **Search and ranking**: To improve recommendations based on natural language queries.
- **Content analysis**: To identify patterns in unstructured text (e.g., summarizing product descriptions or user reviews).

Q9. What is a Model-Based Collaborative approach?
A **model-based collaborative approach** builds a predictive model from the historical data of user-item interactions. Techniques like **matrix factorization** (e.g., SVD) are used to learn latent factors representing users and items. This model can then predict user preferences for new or unseen items.

Q10. Which are used for filtering in a Recommendation Engine?

- **Collaborative Filtering**: Based on user interactions (e.g., ratings, clicks).
- **Content-Based Filtering**: Based on item attributes (e.g., genre, keywords).
- **Hybrid Filtering**: Combines collaborative and content-based methods for improved accuracy.

Q11. For an eCommerce website which one is explicit data?

Explicit data in eCommerce refers to **user-provided feedback**, such as:

- Product ratings
- Written reviews
- User surveys This differs from **implicit data**, such as clicks, views, and time spent on products.

Q12. How would you create a Recommender System for Text Inputs?

To build a recommender system for text inputs:

1. **Text preprocessing**: Use techniques like tokenization, stemming, and vectorization (e.g.,

TF-IDF or Word2Vec) to represent text.

2. **Feature extraction**: Convert text into a numerical format for similarity computations.
3. **Similarity measures**: Use cosine similarity or other distance metrics to find similar content.
4. **Recommendation logic**: Suggest items based on similarity in text features (e.g., descriptions or reviews).

Q13. What are the different methods that you can collect *User Data* for the *Recommendation Process*?

- **Explicit feedback**: Ratings, reviews, or user surveys.
- **Implicit feedback**: Clicks, time spent on a page, purchase history, browsing patterns.
- **Contextual information**: Location, time, device type.
- **User profiles**: Demographic data, preferences, and historical data.

Q14. What are the different types of *Memory-Based Collaborative* approaches?

- **User-Based Collaborative Filtering (UBCF)**: Recommends items by finding users with similar preferences.
- **Item-Based Collaborative Filtering (IBCF)**: Recommends items by finding items that are similar to those the user liked.