Assignment No-5

Title- Develop advanced recommendation systems for e-commerce using matrix factorization algorithm.

Objectives-

- The primary objective of this assignment is to design, implement, and evaluate an advanced recommendation system for an e-commerce platform using matrix factorization algorithms. The system should provide personalized product recommendations to users based on their interactions with products.

Outcomes-

- **Understand** the principles behind recommendation systems and matrix factorization techniques.
- **Implement** a recommendation system using a matrix factorization algorithm (e.g., SVD).
- **Evaluate** the performance of the recommendation system using appropriate metrics.
- **Analyze** the results and understand how the system can be improved.
- **Present** findings and recommendations based on the analysis.

Theory-

1. Introduction to Recommendation Systems

Recommendation systems are essential tools in e-commerce platforms, designed to enhance user experience by providing personalized suggestions. These systems analyze user behavior and preferences to recommend products that are most likely to interest individual users. They can significantly increase user engagement, boost sales, and improve customer satisfaction.

Types of Recommendation Systems:
- Collaborative Filtering: Based on the idea that users with similar tastes will like similar items. This method uses historical interactions among users and items.
- Content-Based Filtering: Recommends items similar to those a user has previously liked, based on item attributes (e.g., features, descriptions).
- Hybrid Methods: Combine collaborative and content-based filtering to leverage the strengths of both approaches, providing more accurate and diverse recommendations.

2. Matrix Factorization

Matrix factorization is a powerful technique used in collaborative filtering. It works by decomposing the user-item interaction matrix into lower-dimensional matrices that represent latent features of users and items. These latent features help capture hidden patterns in user preferences.

Key Concepts:

- User-Item Interaction Matrix: This matrix (often sparse) represents interactions (e.g., ratings, purchases) between users and items. Rows correspond to users, columns correspond to items, and entries represent interactions.

```
 \[   R = \begin{bmatrix}
      r_{11} & r_{12} & \ldots & r_{1n} \\
      r_{21} & r_{22} & \ldots & r_{2n} \\
```

\vdots & \vdots & \ddots & \vdots \\
r_{m1} & r_{m2} & \ldots & r_{mn}
\end{bmatrix}
\]

- Latent Factors: These are hidden features that explain observed ratings. For example, in a movie recommendation system, latent factors could represent genres, while in e-commerce, they could represent product categories or user preferences.

- Decomposition: The interaction matrix \(R \) can be approximated by the product of two lower-dimensional matrices:

\[
R \approx U \times V^T
\]

where \(U \) is the user feature matrix and \(V \) is the item feature matrix.

3. Popular Matrix Factorization Techniques

- Singular Value Decomposition (SVD): A well-known technique that reduces dimensionality by factorizing the user-item matrix into three matrices: user features, singular values, and item features. SVD captures the most significant patterns in the data.

- Alternating Least Squares (ALS): This method iteratively solves for user and item matrices by alternately fixing one and optimizing the other. ALS is particularly useful for large datasets and is effective in implicit feedback scenarios.

- Non-negative Matrix Factorization (NMF): Decomposes the matrix into non-negative factors, making the results easier to interpret. This is beneficial in scenarios where negative values are not meaningful, such as counts or ratings.

4. Model Training and Evaluation

Training a matrix factorization model involves the following steps:

1. Loss Function: The model's performance is typically measured using a loss function, such as Mean Squared Error (MSE):

\[
\text{Loss} = \sum_{(i,j) \in K} (r_{ij} - \hat{r}_{ij})^2
\]

where \(r_{ij} \) is the actual rating and \(\hat{r}_{ij} \) is the predicted rating. The sum runs over all known interactions \(K \).

2. Regularization: To avoid overfitting, regularization techniques are applied to the loss function, penalizing large weights in the latent factor matrices.

3. Gradient Descent: This optimization technique is often used to minimize the loss function, iteratively adjusting the latent factors based on the gradients.

5. Recommendations and User Engagement

Once trained, the model can predict missing ratings for items not yet interacted with by users. Recommendations can be generated by:

- Predicting Scores: Calculate the predicted interaction scores for each user-item pair.
- Top-N Recommendations: For each user, recommend the top-N items with the highest predicted scores.

6. Challenges and Considerations

- Cold Start Problem: New users or items lack sufficient data for making accurate recommendations. Solutions include using demographic data or recommending popular items.
- Sparsity: User-item interaction matrices are often sparse, leading to challenges in training effective models. Techniques like incorporating side information (item features, user demographics) can help.
- Scalability: As datasets grow, ensuring that the recommendation system remains efficient is crucial. Consider distributed systems or optimizations in the matrix factorization algorithms.

Conclusion

Understanding matrix factorization and its application in recommendation systems is crucial for developing effective e-commerce solutions. By leveraging latent factors, we can provide personalized recommendations that enhance user experience and drive sales. This assignment allows students to explore these concepts practically and equips them with the skills needed to implement advanced recommendation systems.

Our CSV doesn’t contain explicit ratings, we’ll convert purchase behavior into a user–item matrix (users × product categories) using Purchase_Amount as implicit rating strength — higher spending = higher “preference”.

Then we’ll apply Matrix Factorization (SVD) to learn latent factors and recommend new product categories for each user.

```python
# Step 1: Import required libraries
import pandas as pd
import numpy as np
from sklearn.decomposition import TruncatedSVD
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import MinMaxScaler

# Step 2: Load dataset
df = pd.read_csv('/content/ecommerce_transactions.csv')

# Step 3: Display first few rows
print("Dataset Preview:\n", df.head())

# Step 4: Select required columns
df = df[['User_Name', 'Age', 'Country', 'Product_Category',
         'Purchase_Amount', 'Payment_Method', 'Transaction_Date']]

# Step 5: Basic preprocessing
df.dropna(inplace=True)

# Normalize purchase amount (scale 0–1)
scaler = MinMaxScaler()
df['Purchase_Score'] = scaler.fit_transform(df[['Purchase_Amount']])

# Step 6: Create User–Item matrix (users × product categories)
user_item_matrix = df.pivot_table(index='User_Name',
                                  columns='Product_Category',
                                  values='Purchase_Score',
                                  aggfunc='mean',
                                  fill_value=0)

print("\nUser–Item Matrix shape:", user_item_matrix.shape)

# Step 7: Apply Matrix Factorization (SVD)
n_components = 8  # latent dimensions
svd = TruncatedSVD(n_components=n_components, random_state=42)
latent_matrix = svd.fit_transform(user_item_matrix)

# Reconstructed matrix = predicted preferences
predicted_matrix = np.dot(latent_matrix, svd.components_)
predicted_df = pd.DataFrame(predicted_matrix,
                            index=user_item_matrix.index,
                            columns=user_item_matrix.columns)

# Step 8: Recommendation Function
def recommend_products(user_name, top_n=5):
    if user_name not in predicted_df.index:
        return "User not found."
    user_predictions = predicted_df.loc[user_name]
    known_items = user_item_matrix.loc[user_name]
    unseen_items = user_predictions[known_items == 0]
    top_recs = unseen_items.sort_values(ascending=False).head(top_n)
    return top_recs

# Step 9: Example usage
sample_user = user_item_matrix.index[0]
recommendations = recommend_products(sample_user, top_n=5)
print(f"\nTop product recommendations for user '{sample_user}':")
print(recommendations)

# Step 10: Evaluate model (simple cosine-similarity sanity check)
user_similarity = cosine_similarity(latent_matrix)
print("\nUser similarity matrix shape:", user_similarity.shape)
print("Example similarity between first two users:",
      round(user_similarity[0, 1], 3))

# Optional: show explained variance ratio
print("\nExplained Variance by SVD components:",
      round(sum(svd.explained_variance_ratio_), 3))
```

```
Dataset Preview:
    Transaction_ID        User_Name  Age  Country Product_Category  \
0               1         Ava Hall   63   Mexico         Clothing
1               2      Sophia Hall   59    India           Beauty
2               3  Elijah Thompson   26   France            Books
3               4     Elijah White   43   Mexico           Sports
4               5       Ava Harris   48  Germany           Beauty

   Purchase_Amount Payment_Method Transaction_Date
0           780.69     Debit Card       2023-04-14
```

```
1          738.56         PayPal       2023-07-30
2          178.34    Credit Card       2023-09-17
3          401.09            UPI       2023-06-21
4          594.83    Net Banking       2024-10-29

User–Item Matrix shape: (100, 8)

Top product recommendations for user 'Ava Allen':
Series([], Name: Ava Allen, dtype: float64)

User similarity matrix shape: (100, 100)
Example similarity between first two users: 0.994

Explained Variance by SVD components: 1.0
```