

Product Recommendation Using Singular Value Decomposition and Cosine Similarity

Farrel Athalla Putra - 13523118¹

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia

¹farrelxag@gmail.com, 13523118@std.stei.itb.ac.id

Abstract— Product recommendation systems have become an essential component of modern digital platforms, enhancing user engagement by delivering personalized suggestions. This paper explores a vector space-based approach to product recommendation by utilizing Singular Value Decomposition (SVD) and Cosine Similarity. Users and items are represented as vectors in a reduced-dimensional space derived from the user-item interaction matrix. Through SVD, the matrix is decomposed to uncover latent features, simplifying complex relationships while maintaining critical patterns. Cosine Similarity measures the alignment between user preferences and item attributes, enabling precise and efficient recommendations. This methodology demonstrates its potential for large-scale applications, such as e-commerce and content platforms, by balancing computational efficiency and recommendation accuracy.

Keywords—Cosine Similarity, product recommendation, Singular Value Decomposition, vector space model.

I. INTRODUCTION

In today's digital age, businesses depend heavily on their ability to keep customers engaged. The more engaged a customer is, the more likely they are to use a product or service, leading to increased revenue. Engagement can come from various activities like browsing, purchasing, or subscribing. When businesses offer products or services that align with customer preferences, it helps build trust and encourages repeat interactions. This is why companies work hard to understand their customers and create experiences that keep them coming back.

One way to improve customer experience is by introducing recommendation systems. These systems suggest items, such as products, services, or content, that users are likely to enjoy based on their preferences or past actions. Recommendation systems are widely used in many industries, from online shopping to entertainment. For example, online stores can recommend products similar to those a user has browsed, and streaming platforms can suggest new movies or music based on viewing or listening habits. These systems not only improve user satisfaction but also help businesses increase sales and engagement.

This paper focuses on building a recommendation system using vector space modeling, a mathematical approach to represent user and item interactions. The method involves Singular Value Decomposition (SVD), which simplifies large datasets by reducing dimensions, and Cosine Similarity, which

measures how closely related items are to user preferences. By applying these techniques, we can create a system that efficiently suggests relevant items to users.

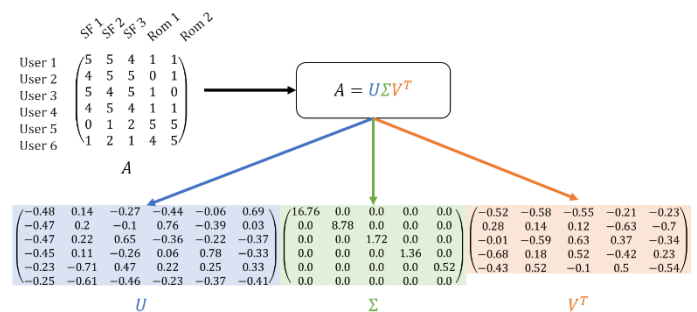


Fig 1.1 SVD in Recommendation System

(Source: <https://yeunun-choo.medium.com/singular-value-decomposition-in-a-movie-recommender-system-e3565ed42066>)

The proposed system offers an effective solution for handling large datasets and delivering accurate recommendations. It can be applied to various platforms, helping businesses create more personalized experiences for their users and improve overall engagement.

II. THEORETICAL BASIS

A. Matrix

Matrix is a rectangular array of numbers arranged in rows and columns. Matrices are an essential data representation widely used in the field of computer science, including graph representation using matrices, digital image representation, kernel matrices in deep learning methods, and matrix representation for systems of linear equations.

Let:

$$A = [a_{ij}], B = [b_{ij}]$$

Then:

The sum of two matrices $C_{m \times n} = A_{m \times n} + B_{m \times n}$ is defined as follows:

$$C = A + B = [c_{ij}], c_{ij} = a_{ij} + b_{ij}, \text{ for } i = 1, \dots, m; j = 1, \dots, n$$

The difference of two matrices $C_{m \times n} = A_{m \times n} - B_{m \times n}$ is defined as follows:

$$C = A - B = [c_{ij}], c_{ij} = a_{ij} - b_{ij}, \text{ for } i = 1, \dots, m; j = 1, \dots, n$$

The sum of two matrices $C_{m \times n} = A_{m \times n} + B_{m \times n}$ is defined as follows:

$$C = A + B = [c_{ij}], c_{ij} = a_{i1}b_{1j} + \dots + a_{in}b_{nj}$$

Condition: matrix multiplication is only defined when the number of columns in A equals the number of rows in B.

The transpose of a matrix $A_{m \times n}$ is a new matrix $A_{n \times m}^T$ obtained by flipping the matrix over its diagonal. In the transpose, the rows of the original matrix become the columns, and the columns become the rows.

$$A^T = [a_{ji}], \text{ for } i = 1, 2, \dots, m; j = 1, 2, \dots, n$$

The determinant is a scalar value that can be computed from the elements of a square matrix. It provides important information about the matrix, such as whether it is invertible.

For a matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, the determinant is:

$$\det(A) = ad - bc$$

For a 3×3 matrix :

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

The determinant is calculated as:

$$\det(A) = a_{11}M_{11} - a_{12}M_{12} + a_{13}M_{13}$$

Where M_{ij} is the determinant of the 2×2 minor matrix obtained by removing the i -th row and j -th column.

For larger matrices, the determinant is calculated using expansion by minors recursively.

Formula for $n \times n$:

$$\det(A) = \sum_{j=1}^n (-1)^{1+j} a_{1j} M_{1j}$$

Where M_{1j} is the determinant of the $(n - 1) \times (n - 1)$ minor matrix obtained by removing the first row and j -th column.

B. Linear System of Equations

A system is considered linear if the highest power of its variables is 1. A system of linear equations with m equations and n variables (x_1, x_2, \dots, x_n) is represented as:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n &= b_m \end{aligned}$$

The system can be expressed in matrix notation as:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

To solve a system of linear equations, Elementary Row Operations (ERO) are applied to an augmented matrix until it is transformed into either Row Echelon Form (REF) or Reduced Row Echelon Form (RREF). There are three types of elementary row operations: multiplying a row by a nonzero constant, swapping two rows, and adding or subtracting a multiple of one

row from another. These operations systematically simplify the matrix, making it easier to solve the system.

$$\begin{bmatrix} 1 & * & * & \dots & * \\ 0 & 1 & * & \dots & * \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

In the Gaussian Elimination Method, the first step is to express the system of equations in augmented matrix form. The augmented matrix is then transformed into row echelon form by applying EROs. In this form, the matrix has a triangular structure with zeros below the diagonal. Once in row echelon form, the system is solved using back substitution, starting with the last row and working upward to find the values of all variables.

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

The Gauss-Jordan Elimination Method goes a step further by transforming the augmented matrix into reduced row echelon form, where each pivot is equal to 1, and all other entries in the pivot column are zeros. In this form, the solution can be directly read from the matrix without requiring back substitution. Gauss-Jordan Elimination is particularly useful when finding the inverse of a matrix or obtaining exact solutions.

C. Cosine Similarity

Cosine similarity measures the similarity between two vectors $Q = (q_1, q_2, \dots, q_n)$ and $D = (d_1, d_2, \dots, d_n)$. It calculates the cosine of the angle θ between the vectors in a multi-dimensional space. This similarity metric is derived from the dot product of the two vectors and their magnitudes. The formula for cosine similarity is:

$$\text{sim}(Q, D) = \cos \theta = \frac{Q \cdot D}{\|Q\| \|D\|}$$

The dot product $Q \cdot D$ is defined as:

$$Q \cdot D = q_1d_1 + q_2d_2 + \dots + q_nd_n = \sum_{i=1}^n q_i d_i$$

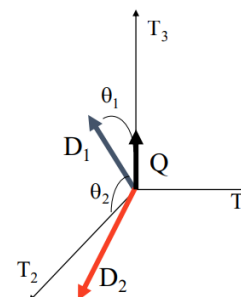


Fig 2.1 Cosine Similarity in Graph Similarity

(Source:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-14-Aplikasi-dot-product-pada-IR-2023.pdf>)

If $\cos \theta = 1$, it means the angle $\theta = 0^\circ$, indicating that the vectors Q and D are perfectly aligned, or pointing in the same direction. In the context of information retrieval, this signifies that the document D is an exact match for the query Q . Larger cosine values, closer to 1, indicate a higher degree of similarity between the vectors, suggesting that the document is more relevant to the query. Conversely, if $\cos \theta = 0$, the vectors Q and D are orthogonal, meaning there is no similarity between them. Finally, if $\cos \theta = -1$, it implies that the vectors point in opposite directions, indicating complete dissimilarity.

C. Eigen Values

If A is an $n \times n$ matrix, a nonzero vector x in \mathbb{R}^n is called an eigenvector of A if the matrix-vector multiplication Ax results in scalar multiple of x . This relationship is written as:

$$Ax = \lambda x$$

Here, λ is called eigenvalue corresponding to the eigenvector x . The operation $Ax = \lambda x$ causes the vector x to shrink or stretch by a factor of λ , with the same direction if λ is positive and the opposite direction if λ is negative.

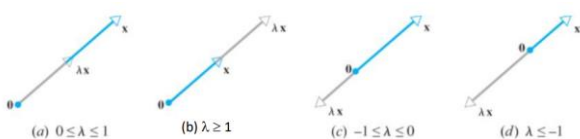


Fig 2.2 Effect of Eigenvalues (λ) on the Scaling and Direction of Eigenvectors

(Source:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-19-Nilai-Eigen-dan-Vektor-Eigen-Bagian1-2023.pdf>)

To calculate eigenvalues and eigenvectors of a matrix A , start with the equation $Ax = \lambda x$, where x is a nonzero vector (the eigenvector) and λ is the eigenvalue. This equation can be rewritten as $(\lambda I - A)x = 0$, where I is the identity matrix of the same size as A . For this equation to have a non-trivial solution ($x \neq 0$), the determinant of $(\lambda I - A)$ must be zero. This leads to the characteristic equation, $\det(\lambda I - A)x = 0$, which is used to find the eigenvalues. The roots of this polynomial equation are the eigenvalues of A . Once the eigenvalues are determined, each eigenvalue λ can be substituted back into the equation $(\lambda I - A)x = 0$ to solve for the corresponding eigenvector x .

D. Singular Value Decomposition (SVD)

For non-square matrices, specifically $m \times n$ matrices, the main diagonal of the matrix is defined as the line starting from the top-left corner and extending downward as far as possible within the matrix.

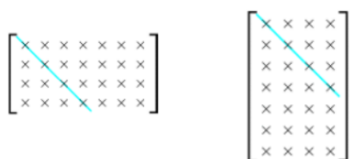


Fig 2.3 Main Diagonal of Non-Square Matrices

(Source:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-21-Singular-value-decomposition-Bagian1-2023.pdf>)

Let A be a $m \times n$ matrix. If $\lambda_1, \lambda_2, \dots, \lambda_n$ are the eigenvalues of $A^T A$, then:

$$\sigma_1 = \sqrt{\lambda_1}, \sigma_2 = \sqrt{\lambda_2}, \dots, \sigma_n = \sqrt{\lambda_n}$$

These values, $\sigma_1, \sigma_2, \dots, \sigma_n$, are called the singular values of the matrix A .

If A is an $m \times n$ matrix of rank k , then A can be factored as:

$$A = U\Sigma V^T = \begin{bmatrix} u_1 & u_2 & \dots & u_k & u_{k+1} & \dots & u_m \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \dots & 0 & & & \\ 0 & \sigma_2 & \dots & 0 & & & \\ \vdots & \vdots & \ddots & \vdots & & & \\ 0 & 0 & \dots & \sigma_k & & & \\ & & & 0_{(m-k) \times k} & & & \\ & & & & 0_{(m-k) \times (n-k)} & & \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_k^T \\ v_{k+1}^T \\ \vdots \\ v_n^T \end{bmatrix}$$

In which U , Σ , and V have sizes $m \times m$, $m \times n$, and $n \times n$, respectively, and in which:

- $V = [v_1 \ v_2 \ \dots \ v_n]$ orthogonally diagonalizes $A^T A$.
- The nonzero diagonal entries of Σ are $\sigma_1 = \sqrt{\lambda_1}$, $\sigma_2 = \sqrt{\lambda_2}$, \dots , $\sigma_k = \sqrt{\lambda_k}$, where $\lambda_1, \lambda_2, \dots, \lambda_k$ are the nonzero eigenvalues of $A^T A$ corresponding to the column vectors of V .
- The column vectors of V are ordered so that $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_k > 0$.
- $u_i = \frac{Av_i}{\|Av_i\|} = \frac{1}{\sigma_i} Av_i$ ($i = 1, 2, \dots, k$).
- $\{u_1, u_2, \dots, u_k\}$ is an orthonormal basis for $\text{col}(A)$.
- $\{u_1, u_2, \dots, u_k, u_{k+1}, \dots, u_m\}$ is an extension of $\{u_1, u_2, \dots, u_k\}$ to an ortho-normal basis for R^m .

III. APPROACH AND METHODOLOGY

The recommendation system implemented in this study is based on the combination of Singular Value Decomposition (SVD) and Cosine Similarity to personalize recommendations for users. The primary goal is to uncover latent patterns in user-item interactions and provide suggestions that align with user preferences. The methodology leverages matrix factorization and similarity measurement techniques, supported by evaluation metrics to assess performance.

To begin, the user-item interaction matrix serves as the foundation. This matrix represents interactions between users (rows) and items (columns), such as ratings, clicks, or purchases.

$$\text{User - Item Matrix} = \begin{bmatrix} 5 & 3 & 0 & 1 & \dots & a_{1n} \\ 4 & 0 & 0 & 1 & \dots & a_{2n} \\ 1 & 1 & 0 & 5 & \dots & a_{3n} \\ 0 & 0 & 4 & 4 & \dots & a_{4n} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & a_{m4} & \dots & a_{mn} \end{bmatrix}$$

Missing values (e.g., 0s) indicate that no interaction has occurred, and these values need to be inferred during the recommendation process. The SVD technique decomposes the user-item matrix into three components: U , Σ , and V^T . U captures user preferences, Σ holds singular values representing the importance of latent features, and V^T represents item attributes. This decomposition reduces the dimensionality of the

data, retaining only the most significant features, which simplifies computations and removes noise.

Once the decomposition is complete, users and items are represented as vectors in the reduced-dimensional space. Cosine Similarity is then applied to measure the closeness between user and item vectors. If a user's preferences are like another user, the system can recommend items that the similar user has interacted with. If items are similar, the system can recommend similar items to what a user has already interacted with.

$$\text{Cosine Similarity} = \frac{u \cdot v}{\|u\| \|v\|}$$

Where u and v represent the vectors for a user and an item, respectively.

Additionally, Mean Squared Error (MSE) assesses the accuracy of predicted ratings compared to actual ratings:

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

where y_i is the actual rating, and \hat{y}_i is the predicted rating. These metrics collectively provide a comprehensive understanding of the system's performance, balancing accuracy and relevance.

IV. RECOMMENDATION SYSTEM

First, create a sample user-item matrix to represent user-item interactions, where each row corresponds to a user, and each column represents an item. The values in the matrix denote the ratings given by users to items, with a value of zero indicating that the user has not rated the item. The purpose of this sample matrix is to provide an example dataset for testing and visualizing the recommendation system. In real-world applications, this matrix would be replaced by actual user-product interaction data from the respective domain.

```

1 def create_sample_data(self):
2     "Sample User-Item Matrix"
3     ratings = [
4         [3.0, 4.0, 5.0, 0.0, 0.0, 5.0, 4.0, 0.0, 3.0, 3.0, 0.0, 4.0, 4.0],
5         [0.0, 4.0, 0.0, 0.0, 3.0, 4.0, 3.0, 4.0, 0.0, 0.0, 4.0, 0.0, 5.0],
6         [5.0, 5.0, 0.0, 0.0, 3.0, 4.0, 0.0, 0.0, 4.0, 5.0, 4.0, 0.0, 5.0],
7         [4.0, 5.0, 4.0, 5.0, 3.0, 3.0, 0.0, 0.0, 0.0, 3.0, 5.0, 3.0, 0.0],
8         [0.0, 3.0, 0.0, 4.0, 3.0, 3.0, 2.0, 0.0, 3.0, 3.0, 4.0, 3.0, 0.0],
9         [0.0, 3.0, 0.0, 3.0, 0.0, 3.0, 2.0, 2.0, 4.0, 3.0, 2.0, 0.0, 3.0],
10        [2.0, 4.0, 0.0, 0.0, 3.0, 2.0, 0.0, 4.0, 4.0, 4.0, 0.0, 3.0, 4.0],
11        [3.0, 3.0, 0.0, 0.0, 4.0, 3.0, 0.0, 3.0, 0.0, 3.0, 0.0, 0.0, 2.0],
12        [0.0, 0.0, 0.0, 0.0, 2.0, 1.0, 0.0, 1.0, 3.0, 0.0, 2.0, 2.0, 1.0],
13        [0.0, 2.0, 1.0, 0.0, 1.0, 2.0, 0.0, 0.0, 2.0, 3.0, 2.0, 1.0, 0.0],
14        [3.0, 2.0, 0.0, 0.0, 2.0, 3.0, 2.0, 3.0, 2.0, 0.0, 1.0, 0.0, 2.0],
15        [0.0, 3.0, 3.0, 1.0, 0.0, 1.0, 0.0, 2.0, 3.0, 2.0, 0.0, 0.0, 1.0],
16    ]
17    return np.array(ratings)

```

Fig 4.1 Implementation of Sample User-Item Matrix

Then, we process the matrix using Singular Value Decomposition (SVD) via eigenvalue decomposition to reduce the dimensionality of the matrix while preserving the most significant patterns of user-item interactions. By focusing on the most important singular values and their corresponding singular vectors, we can approximate the original matrix while filtering out noise or less relevant details.

```

1 def svd(self, A, max_iter=100, tol=1e-10):
2     """
3     SVD implementation using eigenvalue decomposition
4     A = U Σ V^T where:
5     - U: left singular vectors from eigenvectors of AA^T
6     - Σ: square root of eigenvalues
7     - V: right singular vectors from eigenvectors of A^TA
8     """
9     m, n = A.shape
10    k = self.k_factors
11
12    # Center the matrix
13    mask = A != 0
14    row_means = A.sum(axis=1) / mask.sum(axis=1)
15    A_centered = A.copy()
16    for i in range(m):
17        A_centered[i, mask[i]] -= row_means[i]
18
19    # A^T A and AA^T
20    ATA = np.dot(A_centered.T, A_centered) # For V
21    AAT = np.dot(A_centered, A_centered.T) # For U
22
23    # Eigen Decomposition
24    def eigen_decomposition(matrix, k):
25        n = matrix.shape[0]
26        eigenvectors = np.zeros((n, k))
27        eigenvalues = np.zeros(k)
28
29        # Initial matrix
30        M = matrix.copy()
31
32        for i in range(k):
33            vector = np.random.randn(n)
34            vector = vector / np.linalg.norm(vector)
35
36            # Find eigenvector
37            for _ in range(max_iter):
38                new_vector = np.dot(M, vector)
39                norm = np.linalg.norm(new_vector)
40
41                if norm < tol:
42                    break
43
44                new_vector = new_vector / norm
45
46            # Check convergence
47            if np.abs(np.dot(new_vector, vector)) > 1 - tol:
48                break
49
50            vector = new_vector
51
52        # Calculate eigenvalue
53        eigenvalue = np.dot(np.dot(vector.T, M), vector)
54
55        # Store results
56        eigenvectors[:, i] = vector
57        eigenvalues[i] = eigenvalue
58
59        # Deflate matrix
60        M = M - eigenvalue * np.outer(vector, vector)
61
62    return eigenvalues, eigenvectors
63
64    # Get eigenvalues and eigenvectors
65    eigenvalues_ATA, V = eigen_decomposition(ATA, k)
66    eigenvalues_AAT, U = eigen_decomposition(AAT, k)
67
68    # Calculate singular values
69    sigma = np.sqrt(np.abs(eigenvalues_ATA))
70
71    # Ensure U and V are orthogonal
72    U = np.dot(A_centered, V)
73    for i in range(k):
74        if sigma[i] > tol:
75            U[:, i] = U[:, i] / sigma[i]
76
77    # Sort singular values and vectors in descending order
78    idx = np.argsort(sigma)[::-1]
79    sigma = sigma[idx]
80    U = U[:, idx]
81    V = V[:, idx]
82
83    return U, sigma, V.T, row_means

```

Fig 4.2 Implementation of SVD Using Eigenvalue Decomposition

Next, we calculate the predicted ratings to estimate which products are most suitable for each user. Simultaneously, we compute cosine similarity to measure the similarity between users and items, allowing us to identify patterns in user behavior and item relationships. Finally, we combine all these factors—predicted ratings, user similarity, and item similarity—into a unified recommendation model to generate the final product recommendations specifically for each user.

```

1 def predict_ratings(self, U, sigma, Vt, row_means):
2     """Predict ratings using SVD"""
3     return np.dot(U * sigma, Vt) + row_means.reshape(-1, 1)

```

Fig 4.3 Implementation of Predicting the Ratings

```

1 def cosine_similarity(self, matrix):
2     """Cosine similarity"""
3     norm = np.sqrt(np.sum(matrix ** 2, axis=1))
4     matrix_normalized = matrix / norm[:, np.newaxis]
5     similarity = np.dot(matrix_normalized, matrix_normalized.T)
6     return np.clip(similarity, -1, 1) # Values are between -1 and 1

```

Fig 4.4 Implementation of Cosine Similarity

```

1 def get_combined_recommendations(self, user_id, ratings_matrix, reconstructed_matrix,
2     user_similarity, item_similarity, n_recommendations=5):
3     """Get recommendations combining SVD, user-based, and item-based"""
4     # Get unrated items
5     unrated_items = np.where(ratings_matrix[user_id] == 0)[0]
6
7     recommendations = []
8     for item in unrated_items:
9         # 1. SVD-based rating
10        svd_rating = reconstructed_matrix[user_id, item]
11
12        # 2. User-based rating
13        similar_users = np.argsort(user_similarity[user_id])[::-1][1:4] # Top 3 similar users
14        user_based_ratings = []
15        for similar_user in similar_users:
16            if ratings_matrix[similar_user][item] != 0:
17                user_based_ratings.append(
18                    ratings_matrix[similar_user][item] * user_similarity[user_id][similar_user]
19                )
20        user_based_rating = np.mean(user_based_ratings) if user_based_ratings else svd_rating
21
22        # 3. Item-based rating
23        rated_items = np.where(ratings_matrix[user_id] != 0)[0]
24        item_based_ratings = []
25        for rated_item in rated_items:
26            item_based_ratings.append(
27                ratings_matrix[user_id][rated_item] * item_similarity[item][rated_item]
28            )
29        item_based_rating = np.mean(item_based_ratings) if item_based_ratings else svd_rating
30
31        # Combine ratings (weighted average)
32        combined_rating = (0.4 * svd_rating +
33                          0.3 * user_based_rating +
34                          0.3 * item_based_rating)
35
36        recommendations.append((item, combined_rating))
37
38        # Sort and return top N recommendations
39        recommendations.sort(key=lambda x: x[1], reverse=True)
40        return recommendations[:n_recommendations]

```

Fig 4.5 Implementation of Combining All Recommendation Factors

```

1 def main():
2     # Initialize system
3     rec_sys = RecommendationSystem(k_factors=4)
4
5     # Generate sample matrix
6     ratings_matrix = rec_sys.create_sample_data()
7     print("\nUser-Item Matrix : ")
8     print(pd.DataFrame(ratings_matrix).round(2))
9
10    # Perform SVD
11    U, sigma, Vt, row_means = rec_sys.svd(ratings_matrix)
12    print("\nSingular Values:", sigma.round(3))
13    print("\nU Matrix:\n", pd.DataFrame(U).round(2))
14    print("\nVt Matrix:\n", pd.DataFrame(Vt).round(2))
15
16    # Reconstruct matrix and calculate predictions
17    reconstructed_matrix = rec_sys.predict_ratings(U, sigma, Vt, row_means)
18    print("\n Prediction Matrix:\n", pd.DataFrame(reconstructed_matrix).round(2))
19
20    # Calculate similarities
21    user_similarity = rec_sys.cosine_similarity(ratings_matrix)
22    item_similarity = rec_sys.cosine_similarity(ratings_matrix.T)
23
24    # Plot similarity heatmaps
25    plt.figure(figsize=(12, 5))
26
27    plt.subplot(1, 2, 1)
28    sns.heatmap(user_similarity, annot=True, fmt='.2f', cmap='coolwarm')
29    plt.title("User Similarity Matrix")
30
31    plt.subplot(1, 2, 2)
32    sns.heatmap(item_similarity, annot=True, fmt='.2f', cmap='coolwarm')
33    plt.title("Item Similarity Matrix")
34
35    plt.tight_layout()
36    plt.show()
37
38    # Get recommendations for User 1
39    user_id = 1
40    recommendations = rec_sys.get_combined_recommendations(
41        user_id, ratings_matrix, reconstructed_matrix,
42        user_similarity, item_similarity
43    )
44
45    print(f"\nTop 5 Recommendations for User {user_id}:")
46    for item, rating in recommendations:
47        print(f"Item {item}: Predicted rating {rating:.2f}")
48
49    # Calculate MSE for non-zero entries
50    mask = ratings_matrix != 0
51    mse = mean_squared_error(ratings_matrix[mask], reconstructed_matrix[mask])
52    print(f"\nMSE: {mse:.3f}")

```

Fig 4.6 Implementation of Main Program

```

1 Original Ratings Matrix:
2 0 1 2 3 4 5 6 7 8 9 10 11 12
3 0 3.0 4.0 5.0 0.0 0.0 5.0 4.0 0.0 3.0 3.0 0.0 4.0 4.0
4 1 0.0 4.0 0.0 0.0 3.0 4.0 3.0 4.0 0.0 0.0 4.0 0.0 5.0
5 2 5.0 5.0 0.0 0.0 3.0 4.0 0.0 0.0 4.0 5.0 4.0 0.0 5.0
6 3 4.0 5.0 4.0 5.0 3.0 3.0 0.0 0.0 0.0 3.0 5.0 3.0 0.0
7 4 0.0 3.0 0.0 4.0 3.0 3.0 2.0 0.0 3.0 3.0 4.0 3.0 0.0
8 5 0.0 3.0 0.0 3.0 0.0 3.0 2.0 2.0 4.0 3.0 2.0 0.0 3.0
9 6 2.0 4.0 0.0 0.0 3.0 2.0 0.0 4.0 4.0 4.0 0.0 3.0 4.0
10 7 3.0 3.0 0.0 0.0 4.0 3.0 0.0 3.0 0.0 3.0 0.0 0.0 2.0
11 8 0.0 0.0 0.0 0.0 2.0 1.0 0.0 1.0 3.0 0.0 2.0 2.0 1.0
12 9 0.0 2.0 1.0 0.0 1.0 2.0 0.0 0.0 2.0 3.0 2.0 1.0 0.0
13 10 3.0 2.0 0.0 0.0 2.0 3.0 2.0 3.0 2.0 0.0 1.0 0.0 2.0
14 11 0.0 3.0 3.0 1.0 0.0 1.0 0.0 2.0 3.0 2.0 0.0 0.0 1.0
15
16 Singular Values:
17 [3.566 3.248 2.786 2.259]
18
19 U Matrix:
20 0 1 2 3
21 0 -0.30 0.12 0.45 -0.45
22 1 0.12 0.39 -0.08 -0.03
23 2 0.21 0.38 -0.28 -0.20
24 3 0.48 0.24 0.55 -0.15
25 4 0.12 0.17 0.27 0.44
26 5 0.11 -0.11 -0.32 0.22
27 6 0.52 0.04 -0.11 -0.41
28 7 -0.11 -0.34 0.10 0.13
29 8 0.24 -0.43 -0.03 0.28
30 9 0.21 0.20 -0.34 0.18
31 10 -0.30 0.12 -0.30 -0.10
32 11 0.35 -0.49 -0.01 -0.43
33
34 Vt Matrix:
35 0 1 2 3 4 5 6 7 8 9 10 11 \
36 0 -0.13 0.41 -0.02 0.09 -0.31 -0.62 -0.08 -0.04 0.41 0.16 0.27 -0.20
37 1 0.06 0.03 -0.15 0.27 -0.54 0.21 -0.14 0.17 -0.43 0.05 0.06 -0.16
38 2 -0.22 0.10 0.29 0.28 0.14 -0.05 0.05 -0.01 -0.31 -0.60 0.53 -0.07
39 3 0.34 -0.29 -0.46 0.46 0.15 0.08 -0.30 -0.32 0.19 0.04 0.31 -0.07
40
41 Prediction Matrix:
42 0 1 2 3 4 5 6 7 8 9 10 11 \
43 0 3.43 3.87 4.70 3.78 4.04 4.49 4.29 4.30 2.71 2.93 3.96 4.04
44 1 3.91 4.07 3.62 4.15 2.99 3.87 3.65 4.09 3.54 4.12 3.91 3.59
45 2 4.37 4.76 4.16 4.34 3.31 4.18 4.24 4.72 4.30 5.00 4.10 4.12
46 3 3.49 4.67 4.03 4.84 3.19 2.94 3.62 3.83 3.84 3.30 5.32 3.29
47 4 3.26 3.09 2.78 3.97 2.94 3.00 2.74 2.86 3.01 2.80 3.97 2.82
48 5 3.07 2.70 2.32 2.70 2.79 2.55 2.60 2.56 3.46 3.39 2.55 2.78
49 6 2.85 4.34 3.61 3.02 2.50 2.15 3.42 3.59 3.95 3.78 3.40 3.02
50 7 3.02 2.75 3.11 2.88 3.79 3.02 3.11 2.73 3.28 2.74 3.06 3.21
51 8 1.74 1.85 1.59 1.68 2.27 0.94 1.65 1.24 2.80 1.87 2.02 1.72
52 9 2.04 1.86 1.16 1.91 1.10 1.52 1.43 1.72 2.14 2.49 1.62 1.53
53 10 2.49 1.78 2.05 1.90 2.20 2.98 2.28 2.41 1.84 2.56 1.45 2.45
54 11 1.41 2.74 2.65 1.24 2.30 0.82 2.40 1.99 3.00 2.09 1.94 2.06

```

Fig 4.7 Result of SVD Calculation



Fig 4.8 Heatmaps, Recommendation Results, and MSE

The heatmaps above display the similarity matrices calculated using cosine similarity for both users and items in the recommendation system. The User Similarity Matrix (left) highlights how closely each user relates to others based on their interaction with items. Values closer to 1 (red) indicate higher similarity, signifying that the users have similar preferences, while values closer to 0 or negative (blue) indicate low or no similarity. Similarly, the Item Similarity Matrix (right) illustrates how closely items are related to each other based on users' preferences and interactions.

These similarity measures were then used to enhance the

prediction of recommendations by identifying related users and items to infer ratings. In the final recommendation results, the system calculated the predicted ratings for the unrated items using Singular Value Decomposition (SVD) and combined it with similarity metrics to provide recommendations. For instance, the system identified Item 11, Item 9, and Item 2 as the top recommendations for User 1 with predicted ratings of 3.10, 3.05, and 2.96, respectively. This outcome results from considering not only the reconstructed rating matrix but also leveraging the relationships between users and items to provide a balanced and accurate recommendation.

Because the Mean Squared Error (MSE) is small, it demonstrates that the method of product recommendation using Singular Value Decomposition (SVD) combined with cosine similarity is one of the effective ways to predict user preferences accurately. A low MSE indicates that the predicted ratings are very close to the actual ratings, which validates the robustness and reliability of this approach.

V. CONCLUSION

Product Recommendation System using Singular Value Decomposition (SVD) and cosine similarity is one of many approaches used in modern recommendation systems to provide accurate and personalized suggestions. Compared to traditional methods, this combination offers enhanced precision by decomposing the user-item interaction matrix into its latent factors, which represent hidden patterns in user preferences and item characteristics. The use of cosine similarity further complements this by measuring the relationships between users or items, allowing for more relevant recommendations.

While there are more advanced methods such as deep learning models or hybrid filtering systems that could provide additional accuracy, SVD with cosine similarity is sufficient for small to medium-scale applications due to its simplicity and computational efficiency. This approach effectively balances complexity and accuracy, making it a reliable solution for personalized recommendations in various industries, such as e-commerce, entertainment, and education platforms.

VI. APPENDIX

The source code used to implement the product recommendation system using SVD and cosine similarity:

<https://github.com/farrelathalla/Product-Recommendation-with-SVD-and-Cosine-Sim.git>

VII. ACKNOWLEDGMENT

The author wishes to express gratitude, first and foremost, to Allah SWT for the guidance provided throughout the learning process and the writing of this paper. Appreciation is also extended to the lecturers of ITB Linear Algebra and Geometry IF2123, Mr. Rinaldi Munir and Mr. Rila Mandala, for imparting their knowledge and guiding the students during the course. Additionally, the author is deeply thankful to family and friends for their unwavering support throughout the semester.

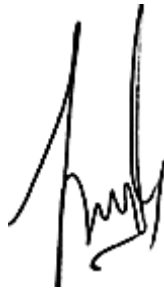
REFERENCES

- [1] Y. Koren, R. Bell, and C. Volinsky, "Matrix Factorization Techniques for Recommender Systems," *IEEE Computer Society*, vol. 42, no. 8, 2009.
- [2] H. Polat and Wenliang Du, "SVD-based Collaborative Filtering with Privacy-Preserving," *Expert Systems with Applications*, vol. 67, 2017.
- [3] Munir, Rinaldi. "<http://informatika.stei.itb.ac.id/~rinaldi.munir/>"
- [4] <https://yeunun-choo.medium.com/singular-value-decomposition-in-a-movie-recommender-system-e3565ed42066>

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 Desember 2024



Farrel Athalla Putra - 13523118