Behind Netflix's Recommendations: The Influence of Singular Value Decomposition (SVD)

Varel Tiara and 13523008
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13523008@std.stei.itb.ac.id, vareltiara@gmail.com

***Abstract*—This paper explores the implementation and significance of Singular Value Decomposition (SVD) in Netflix's recommendation system, focusing on its evolution from basic SVD to advanced variants like Funk-SVD and SVD++. The study examines how these algorithms address the challenges of data sparsity and personalization in large-scale entertainment platforms. Through detailed analysis of the Netflix Prize dataset, we demonstrate the practical implementation of SVD-based recommendation systems, including data preprocessing, model training, and evaluation. The research shows how SVD++ enhances recommendation accuracy by incorporating both explicit and implicit user feedback, leading to more personalized content suggestions. Our implementation results, measured through RMSE and MAE metrics, indicate significant improvements in prediction accuracy when compared to traditional recommendation approaches. The findings highlight how SVD-based techniques effectively handle the complexity of user-item interactions in modern streaming platforms, contributing to an enhanced user experience through more relevant content recommendations.**

Keywords—Recommendation Systems, Singular Value Decomposition (SVD), SVD++, Netflix, Matrix Factorization, Collaborative Filtering, Personalization, Data Sparsity, User-Item Matrix

I. INTRODUCTION

Amid the rapid wave of digitalization, recommendation systems have emerged as one of the most significant innovations in delivering personalized experiences to users. In the entertainment industry, particularly on movie platforms like Netflix, the biggest challenge is not just providing users with movies but rather presenting truly relevant choices from an almost infinite array of options. These recommendation systems help users discover engaging content and strengthen their loyalty to platforms like Netflix. For instance, when users log into a platform and are presented with intriguing suggestions, their curiosity often compels them to watch. As a result, recommendation systems not only provide more relevant content but also create an addictive experience that keeps users returning.

Behind the scenes, advanced algorithms act as "invisible guides" that uncover unique patterns in user behavior. One of the most powerful approaches utilized is Singular Value Decomposition (SVD), a matrix decomposition technique that maps potential relationships between users and content. By breaking down user-item matrices into smaller-dimensional components, SVD reveals hidden arrangements and unique properties of the content. The result is recommendations that are not only precise but also feel highly personalized for everyone.

The key strength of SVD lies in its ability to tackle the challenge of data sparsity, where most of the input data consists of empty or missing values. For a large-scale platform like Netflix, which has billions of users and trillions of interactions, sparse data poses a significant hurdle. By leveraging its capabilities, SVD can fill these gaps with accurate predictions and derive meaningful insights into user preferences, even with limited data.

Netflix, one of the leading companies in the digital entertainment industry, heavily relies on recommendation systems to deliver an intuitive and enjoyable user experience. A primary application of SVD on Netflix is on the homepage, where users are presented with a curated list of movies that seem "tailored just for them." The algorithm integrates various variables, such as watch history, interaction patterns, and implicit preferences, to craft recommendations that capture the user's attention right from the start.

This paper aims to delve deeper into how SVD serves as a cornerstone of Netflix's recommendation system. From its mathematical foundations to practical implementation, the discussion will provide insights into how this algorithm enhances content relevance and revolutionizes the way we experience entertainment in the digital era.

II. BASIC THEORY

A. Recommendation Systems

A recommendation system is an information filtering technology designed to help users discover relevant items based on their preferences. These items can include movies, songs, videos, or other products. Recommendation systems play a crucial role in various digital platforms such as e-commerce, streaming services, social media, and other applications that emphasize personalized experiences. The system analyzes user data, such as browsing history, viewing history, interactions, or positively rated movies, to identify patterns and trends that provide relevant recommendations to users. Examples of recommendation system applications include:

1. E-commerce platforms like Amazon: recommending items based on browsing and purchase history.
2. Music streaming services like Spotify: suggesting songs or artists based on listening history.
3. Podcast streaming providers such as Netflix recommend movies and TV series based on your watching history.

Recommendation systems can be classified into several types, each with its approach to suggesting content. One of the most widely used methods is Collaborative Filtering, which recommends items based on evaluating user interactions and finding similarities between users (user-based) or items (item-based). In the context of movies, if two users enjoy the same film, the system can recommend movies liked by one user to the other.

1. User-Based Collaborative Filtering: This method recommends movies based on the similarity between users. For example, if two users often watch action movies, a movie watched by one user can be recommended to the other.
2. Item-Based Collaborative Filtering: This method recommends movies based on the similarity between items. For instance, if a user watches a movie about a superhero, the system may recommend other superhero movies or movies in a similar genre.

Another common method is Content-Based Filtering, which recommends items based on the attributes or characteristics of content that the user has liked before. For example, if a user enjoys action movies, the system will suggest more action films based on similar genres or keywords.

Hybrid Systems combine both Collaborative Filtering and Content-Based Filtering to generate more accurate and diverse recommendations. For example, Netflix uses both viewing data (collaborative) and movie features like genre and actors (content-based) to suggest movies.

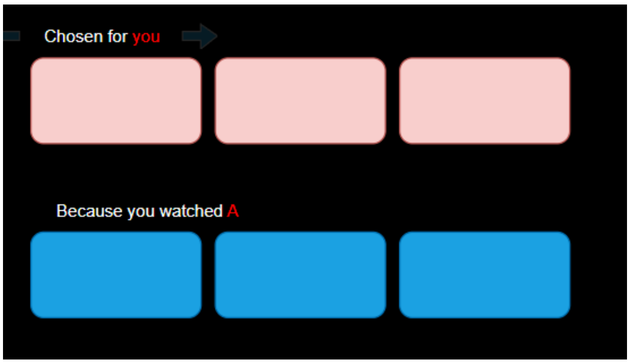


Image 1. Recommender Systems
(source: writer's archive)

Collaborative filtering: content is selected based on user's historical feedback.
Content filtering: content is selected based on some similarity metrics with A.

In movie recommendations, deep learning models are increasingly applied to understand complex data such as movie metadata, user behavior, and visual/audio elements in the movies. Some commonly used deep learning models for this purpose include:

1. Autoencoders: These models compress user preferences into a latent space and reconstruct recommendations based on discovered patterns.
2. Convolutional Neural Networks (CNN): CNNs analyze visual content, such as movie posters and scenes, to determine the relevance or popularity of a movie.
3. Recurrent Neural Networks (RNN): These networks utilize users' viewing history to provide recommendations based on temporal patterns, such as preferences for specific genres over time.
4. Attention Mechanisms: These mechanisms identify important elements within movies, such as specific actors or themes, to offer more precise recommendations.

Recommendation systems play a crucial role in enhancing the movie viewing experience in multiple ways:

1. Faster Decision Making: Users no longer need to manually search for movies, saving time in finding relevant content.
2. Personalization: Recommendations are more tailored to user interests, providing a personalized movie-watching experience.
3. Increased Engagement: By suggesting movies that align with users' preferences, recommendation systems encourage users to watch more content, increasing the time spent on the platform.

B. Matrix Factorization in Recommendation Systems

Matrix factorization (MF) is a foundational approach in recommendation systems, especially those based on collaborative filtering. This technique deconstructs large, sparse user-item interaction matrices into smaller, dense ones, uncovering latent features that govern user preferences and item characteristics. These latent features capture hidden patterns in the data, enabling systems to predict user preferences for items they have not yet interacted with.

At its core, matrix factorization breaks down the interaction matrix into two smaller matrices: one representing user preferences and the other representing item attributes. These smaller matrices, when multiplied together, approximate the original matrix and fill in missing interactions. For example, if Person 1 and Person 2 enjoy movies B and C while Person 3 only likes movie B, matrix factorization can infer that users who like movie B often enjoy movie C, making C a recommended option for Person 3.

User	Items		
	A	B	C
1	8	10	9
2		10	9
3		9	?

Image 2. User's Ratings Table on Items
(source: writer's archive)

Matrix factorization's ability to reveal underlying relationships makes it particularly effective for personalization. It is widely applied across various domains, such as:

1. Movie Recommendations: Identifying films that align with a user's tastes based on shared preferences with others.
2. E-commerce: Tailoring product suggestions to individual customers based on their browsing and purchase history.
3. Streaming Platforms: Recommending music or movies by analyzing patterns in user behavior.

One of the key advantages of matrix factorization is its capacity to handle large, sparse datasets, a common feature of real-world recommendation systems. By extracting latent features, it provides accurate and scalable solutions to personalization challenges, enhancing user experiences across diverse applications.

C. Singular Value Decomposition (SVD) in Recommendation Systems

Singular Value Decomposition (SVD) is a matrix factorization technique used to decompose large matrices into smaller, more manageable components. It is a mathematical method that allows breaking down a matrix into its core components—singular values and vectors—which can then be used to make predictions about the original matrix. In the context of recommendation systems, SVD is applied to analyze large and sparse user-item interaction matrices, where each row represents a user, each column represents an item, and the cells contain ratings or interactions between users and items.

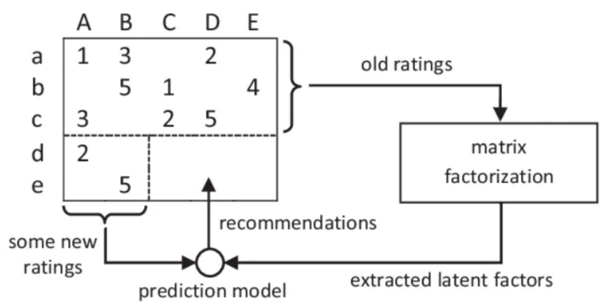


Image 3. SVD in Recommendation System
(https://medium.com/@ritik_gupta/how-singular-value-decomposition-svd-is-used-in-recommendation-systems-clearly-explained-201b24e175db)

A user-item interaction matrix, such as the one used in movie recommendation systems like Netflix, represents how much each user likes or interacts with an item (e.g., a movie). Most entries in this matrix are typically empty because users only rate or interact with a small fraction of the available items. By applying SVD, this large matrix is decomposed into three smaller matrices: one representing user preferences, another representing item attributes, and a diagonal matrix containing singular values that represent the strength of these latent factors.

Mathematically, SVD decomposes the matrix A (user-item matrix) into three components:

$$A = U\sum V^T$$

Where:

- A is the matrix (m x n matrix)
- U is the matrix of latent factors for users (m x m matrix),
- $\sum$ is a diagonal matrix containing the singular values, which represent the strength of the latent factors (m x n matrix),
- V^T is the matrix of latent factors for items (n x n matrix).

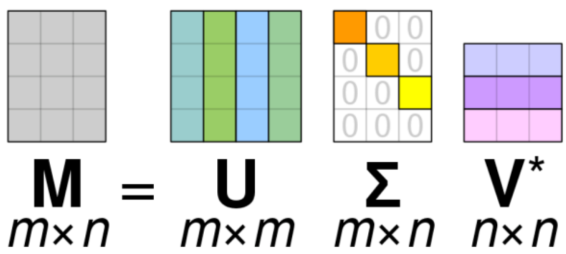


Image 4. Decomposition SVD
(https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-21-Singular-value-decomposition-Bagian1-2023.pdf)

The latent representations of users and items in U and V can then be used to make recommendations in various ways:

1. Finding Similar Users or Items: Similarity between latent representations in U or V can identify users with similar preferences or items with shared characteristics.
2. Predicting Ratings: To predict how a user would rate an unseen item, we can take the dot product of the user's latent representation in U with the item's latent representation in V.

SVD is also particularly effective at handling missing data, a common challenge in recommendation systems. By leveraging the latent representations, SVD can fill in missing ratings and enable more accurate predictions. Additionally, its ability to reduce dimensionality makes it an ideal choice for large-scale recommendation systems.

D. Funk-SVD: The Foundation of Matrix Factorization

Funk-SVD is a modified version of the Singular Value Decomposition (SVD) technique, specifically adapted for recommendation systems. Unlike traditional SVD, which decomposes a matrix into three components, Funk-SVD directly optimizes the user U and item V matrices without explicitly constructing the diagonal matrix Σ. It employs a gradient descent method to minimize an objective function that includes a regularization term, which helps to prevent overfitting by penalizing large values in the latent factors.

Estimated rating matrix:

$$\hat{A} = UV^T$$

Objective function:

$$J = \sum \frac{(A - UV^T)^2}{2} + \frac{\lambda}{2}\left(\sum |U|^2 + \sum |V|^2\right)$$

The Funk-SVD process starts with a minimal number of latent factors and incrementally increases them until the optimal error value is achieved. To find the optimal U and V, the Alternating Least Squares (ALS) procedure is used. This involves alternating between solving for user and item factors while keeping the other fixed, iteratively refining the factors until convergence. Once trained, the U and V matrices can predict missing ratings and generate recommendations by capturing the latent patterns in user preferences and item attributes.

Funk-SVD laid the groundwork for further advancements, including SVD++, which extended the method to incorporate implicit feedback and bias terms. While Funk-SVD is highly effective for explicit rating datasets, it also served as a stepping stone for more sophisticated models in recommendation systems.

E. SVD++: Enhanced Singular Value Decomposition

Building on the foundation of Funk-SVD, SVD++ introduces improvements specifically designed to enhance collaborative filtering (CF) recommendation systems. While Funk-SVD focuses primarily on explicit user ratings, SVD++ incorporates implicit feedback, such as whether a user interacted with an item without providing a rating. This inclusion of implicit interactions helps to better address the challenges of sparse datasets, which are common in recommendation systems.

SVD++ retains the core principles of Funk-SVD while adding terms to account for implicit interactions and bias adjustments for both users and items. By analyzing both explicit and implicit feedback, SVD++ significantly improves the accuracy of recommendations, making it particularly effective in scenarios where user-item interaction data is incomplete or limited.

In recommendation systems, the user-item rating matrix R^{m*n}, where $R_{i,j}$ denotes the rating user i gives to item j, is often highly sparse.

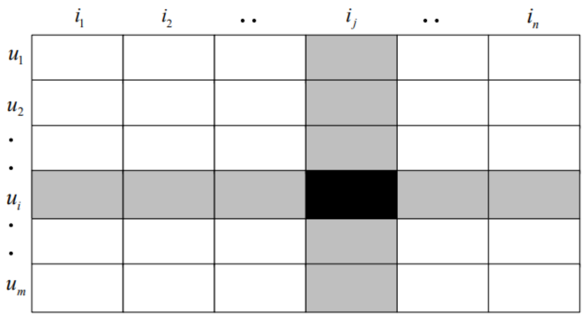


Image 5. R^{m*n} Scoring Matrix
(source: writer's archive)

The rows in the matrix correspond to individual users, while the columns represent different items. This creates a user-item matrix that is typically highly sparse, meaning that only a small portion of the total entries have known ratings, which reflects real-world scenarios. The scoring matrix U can be represented as the product of two smaller matrix.

$$U = \begin{bmatrix} u_{11} & \cdots & u_{1k} \\ \vdots & \ddots & \vdots \\ u_{m1} & \cdots & u_{mk} \end{bmatrix} \times \begin{bmatrix} i_{11} & \cdots & i_{1n} \\ \vdots & \ddots & \vdots \\ i_{k1} & \cdots & i_{kn} \end{bmatrix} = \begin{bmatrix} p_1^T \\ \vdots \\ p_m^T \end{bmatrix} \times [q_1 \cdots q_n]$$

SVD decomposes R into two matrices $P_{m\times k}$ (user factors) and $Q_{k\times n}$ (item factors), such that:

$$R = P \cdot Q^T$$

Here, k represents the number of latent factors. The dot product of the user and item factor vectors, p_u and q_i, predicts the rating $\hat{r}_{ui}$ as:

$$\hat{r}_{ui} = p_u^T q_i$$

To minimize the prediction error, SVD optimizes P and Q by minimizing the squared error between actual and predicted ratings. Regularization terms are added to prevent overfitting, yielding the Regularized SVD (RSVD) model:

$$\sum_{u,i} e_{ui}^2 = \min_{q_i, p_u} \sum_{u,i} \left(r_{ui} - \sum_{k=1}^{K} p_{uk} q_{ki}^T \right)^2$$

To address the overfitting problem in SVD (Singular Value Decomposition), regularization techniques are commonly used. One of the methods is Regularized SVD

(RSVD), which adds a regularization term to the basic SVD formulation to control the complexity of the model and reduce overfitting. The formulation of RSVD is as follows:

$$\sum_{u,i} e_{ui}^2 = \min_{q_i, p_u} \sum_{u,i} \left(r_{ui} - \sum_{k=1}^{K} p_{uk} q_{ki}^T \right)^2 + \frac{\lambda}{2} \sum_{u} |p_u|^2 + \frac{\lambda}{2} \sum_{i} |q_i|^2$$

Where:

- r_{ui} is the observed rating of user u for item i,
- p_u and q_i are the latent factors for users and items,
- λ is the regularization parameter that helps prevent overfitting.

To further improve the performance of RSVD, the algorithm is extended to SVD++ by incorporating implicit feedback, such as user interaction history or item click history. This additional information can enhance the model by providing more context for recommendations. The prediction of rating $\hat{r}_{ui}$ in SVD++ is calculated as:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T \left(p_u + \frac{1}{\sqrt{|N(u)|}} \sum_{j \in N(u)} y_j \right)$$

Where:

- μ is the global average,
- b_i and b_u are the offsets for item i and user u,
- p_u and q_i are the latent factors for item i and user u,
- N(u) is the set of items that user uuu has interacted with,
- y_j is the implicit feedback for item j.

The offsets b_i and b_u are computed as:

$$b_i = \frac{\sum_{u \in R(i)} (r_{ui} - \mu)}{\alpha + |R(i)|}, \quad b_u = \frac{\sum_{i \in R(u)} (r_{ui} - \mu - b_i)}{\beta + |R(u)|}$$

Where R(i) and R(u) represent the sets of users who rated item i and the set of items rated by user u, respectively.

The goal is to minimize the Mean Squared Error (MSE) loss:

$$\min_{i,u} \sum (r_{ui} - \hat{r}_{ui})^2$$

To prevent overfitting, an L2 regularization term is added to the objective function:

$$\min \sum_{i,u} \left[\left(r_{ui} - \mu - b_i - b_u - q_i^T \left(p_u + \frac{1}{\sqrt{|N(u)|}} \sum_{j \in N(u)} y_j \right) \right)^2 + \lambda \left(||b_u||^2 + ||b_i||^2 + ||p_u||^2 + ||q_i||^2 + \sum_{j \in N(u)} ||y_j||^2 \right) \right]$$

To solve this objective, two main optimization methods are commonly used: iterative least squares and gradient descent. Gradient descent is typically preferred for large datasets due to its efficiency. The update rule for gradient descent is as follows:

$$x^{(k+1)} = x^{(k)} - \alpha \nabla f\left(x^{(k)}\right)$$

Where α is the learning rate, and the update for each parameter is:

- $e_{ui} \leftarrow r_{ui} - \hat{r}_{ui}$
- $b_i \leftarrow b_i + \gamma(e_{ui} - \lambda b_i)$
- $b_u \leftarrow b_u + \gamma(e_{ui} - \lambda b_u)$
- $p_u \leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda p_u)$
- $q_i \leftarrow q_i + \gamma\left(e_{ui} \cdot \left(p_u + \frac{1}{\sqrt{|N(u)|}} \sum_{j \in N(u)} y_j\right) - \lambda q_i\right)$
- $y_j \leftarrow y_j + \gamma\left(e_{ui} \cdot \frac{1}{\sqrt{|N(u)|}} \cdot q_i - \lambda y_j\right)$

F. Implementing SVD++ in Netflix

SVD++ is an enhancement to the standard SVD technique that addresses its limitations by incorporating implicit data, such as whether a user interacted with an item, even without explicitly rating it. In SVD++, this implicit data is incorporated by extending the representation of both users and items. The model considers that user interactions with items (whether rated or not) provide additional insights into user preferences.

On Netflix, SVD++ has been used to improve the quality of its recommendation system, especially during the Netflix Prize competition, which aimed to enhance the accuracy of predicting movie ratings on the platform. SVD++ addresses the problem of data sparsity by incorporating implicit feedback, such as whether a user watched a movie or gave a positive rating, which was previously not considered in traditional SVD models. In SVD++, the model considers two key components: the latent factors of users and items, as well as the contribution from implicit interactions that were not accounted for in the standard SVD model.

More specifically, SVD++ adds a user-related factor based on the items they have seen, even if they have not explicitly rated them. This provides a significant advantage by reducing bias towards items with fewer ratings, which often happens with new or less popular items. In the context of Netflix, SVD++ helps generate more accurate recommendations by utilizing all available data, including implicit data.

The process of SVD++ on Netflix begins by identifying both explicit and implicit interactions, and then combining them into a user-item matrix. Machine learning algorithms are then used to optimize two main matrices—the user and item matrices—by minimizing prediction errors in the recommendations. This process is iterative, refining the model with each cycle until the predicted ratings closely match the actual ratings given by users.

The use of SVD++ in the Netflix Prize gave Netflix a significant edge by improving recommendation accuracy, directly impacting user experience. By combining explicit and implicit data, Netflix can provide recommendations that are more personalized and relevant based on more information about user behavior. As a result, while Netflix now employs many advanced techniques, SVD++ remains a key component in delivering a highly tailored viewing experience for each user.

Recommendation systems that use SVD++ can provide more accurate suggestions by considering both types of data—explicit (direct ratings) and implicit (interactions such as watching or clicking). This makes SVD++ especially effective in handling sparse interaction matrices, a common challenge for large platforms like Netflix.

III. Implementation

A. Netflix Competition Data

To build a recommender system, we utilize the Netflix competition dataset. This dataset includes a movie catalog (movie_titles.csv) and user ratings spread across four files (combined_data_x.txt).

1. Data Description and Preprocessing

1.Movie Catalog

The movie catalog is stored in the movie_titles.csv file, containing details such as movie ID, release year, and movie title. The file is loaded into a DataFrame, and the Movie_Id column is set as the index for easier reference. The image shows how the movie data is loaded and indexed.

```python
# Load movie catalog
df_mov_titles = pd.read_csv(path + '/movie_titles.csv',
                            sep=',',
                            header=None,
                            names=['Movie_Id', 'Year', 'Name'],
                            usecols=[0, 1, 2],
                            encoding="ISO-8859-1")
df_mov_titles.set_index('Movie_Id', inplace=True)
```

Image 6. Load Movie Catalog
(source: writer's archive)

2.User Ratings

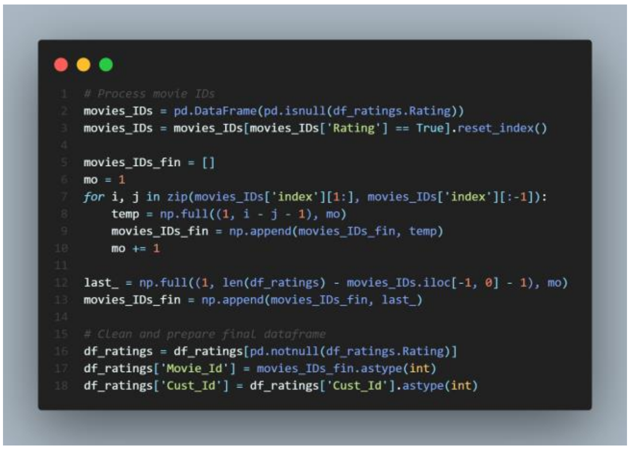
The user ratings are divided into four files. Ratings are loaded, cleaned, and combined into a single DataFrame. Rows containing

```python
# Load ratings files
files = ['combined_data_1.txt', 'combined_data_2.txt',
         'combined_data_3.txt', 'combined_data_4.txt']
df_ratings = pd.DataFrame()

# Only load first file for testing if needed
if sample_size:
    files = files[:1]

for f in files:
    staging = pd.read_csv(path + '/' + f,
                          header=None,
                          names=['Cust_Id', 'Rating'],
                          usecols=[0, 1])
    staging['Rating'] = staging['Rating'].astype(float)

    if len(df_ratings) > 0:
        df_ratings = pd.concat([df_ratings, staging], ignore_index=True)
    else:
        df_ratings = staging
    del staging
```

Image 7. Load User Ratings
(source: writer's archive)

3.Movie ID Assignment

Each rating is associated with its corresponding movie ID by processing the ratings data and identifying boundaries where movie IDs change. This results in a clean DataFrame that contains movie IDs and corresponding user ratings. The image demonstrates how movie IDs are assigned and the data is cleaned.



```python
# Process movie IDs
movies_IDs = pd.DataFrame(pd.isnull(df_ratings.Rating))
movies_IDs = movies_IDs[movies_IDs['Rating'] == True].reset_index()

movies_IDs_fin = []
mo = 1
for i, j in zip(movies_IDs['index'][1:], movies_IDs['index'][:-1]):
    temp = np.full((1, i - j - 1), mo)
    movies_IDs_fin = np.append(movies_IDs_fin, temp)
    mo += 1

last_ = np.full((1, len(df_ratings) - movies_IDs.iloc[-1, 0] - 1), mo)
movies_IDs_fin = np.append(movies_IDs_fin, last_)

# Clean and prepare final dataframe
df_ratings = df_ratings[pd.notnull(df_ratings.Rating)]
df_ratings['Movie_Id'] = movies_IDs_fin.astype(int)
df_ratings['Cust_Id'] = df_ratings['Cust_Id'].astype(int)
```

Image 8. Assign movie IDs and clean data
(source: writer's archive)

2. Data Sampling

To optimize performance, especially for large datasets, a stratified sample of the ratings data is taken to maintain the relative proportions of high and low-rated movies and customers. This step is essential for ensuring that the sample represents the full dataset well, without overwhelming computational resources. The image shows how the data is sampled to maintain proportionality.

```python
# Sample if requested
if sample_size:
    # Stratified sampling to maintain user representation
    user_counts = df_ratings['Cust_Id'].value_counts()
    users_with_min_ratings = user_counts[user_counts >= 5].index
    df_ratings = df_ratings[df_ratings['Cust_Id'].isin(users_with_min_ratings)]
    df_ratings = df_ratings.sample(n=min(sample_size, len(df_ratings)), random_state=42)
```

Image 9. Data Sampling
(source: writer's archive)

```python
def load_and_process_data(path, sample_size=None):
    # Load movie catalog
    df_mov_titles = pd.read_csv(path + '/movie_titles.csv',
                                sep=',',
                                header=None,
                                names=['Movie_Id', 'Year', 'Name'],
                                usecols=[0, 1, 2],
                                encoding="ISO-8859-1")
    df_mov_titles.set_index('Movie_Id', inplace=True)

    # Load ratings files
    files = ['combined_data_1.txt', 'combined_data_2.txt',
             'combined_data_3.txt', 'combined_data_4.txt']
    df_ratings = pd.DataFrame()

    # Only load first file for testing if needed
    if sample_size:
        files = files[:1]

    for f in files:
        staging = pd.read_csv(path + '/' + f,
                              header=None,
                              names=['Cust_Id', 'Rating'],
                              usecols=[0, 1])
        staging['Rating'] = staging['Rating'].astype(float)

        if len(df_ratings) > 0:
            df_ratings = pd.concat([df_ratings, staging], ignore_index=True)
        else:
            df_ratings = staging
        del staging

    # Process movie IDs
    movies_IDs = pd.DataFrame(pd.isnull(df_ratings.Rating))
    movies_IDs = movies_IDs[movies_IDs['Rating'] == True].reset_index()

    movies_IDs_fin = []
    mo = 1
    for i, j in zip(movies_IDs['index'][1:], movies_IDs['index'][:-1]):
        temp = np.full((1, i - j - 1), mo)
        movies_IDs_fin = np.append(movies_IDs_fin, temp)
        mo += 1

    last_ = np.full((1, len(df_ratings) - movies_IDs.iloc[-1, 0] - 1), mo)
    movies_IDs_fin = np.append(movies_IDs_fin, last_)

    # Clean and prepare final dataframe
    df_ratings = df_ratings[pd.notnull(df_ratings.Rating)]
    df_ratings['Movie_Id'] = movies_IDs_fin.astype(int)
    df_ratings['Cust_Id'] = df_ratings['Cust_Id'].astype(int)

    # Sample if requested
    if sample_size:
        # Stratified sampling to maintain user representation
        user_counts = df_ratings['Cust_Id'].value_counts()
        users_with_min_ratings = user_counts[user_counts >= 5].index
        df_ratings = df_ratings[df_ratings['Cust_Id'].isin(users_with_min_ratings)]
        df_ratings = df_ratings.sample(n=min(sample_size, len(df_ratings)), random_state=42)

    return df_ratings, df_mov_titles
```

Image 10. Load and Process Data
(source: writer's archive)

B. Building The Recommendation System

The recommender system is constructed using the surprise library. We leverage the Singular Value Decomposition (SVD) model, which is efficient for explicit feedback scenarios like movie ratings.

1. Model Training and Validation

The dataset is converted into a surprise.Dataset object, which is then used for training. To identify optimal hyperparameters, we conduct a 5-fold cross-validation.

```python
reader = Reader(rating_scale=(1, 5))
    data = Dataset.load_from_df(df_ratings[['Cust_Id', 'Movie_Id', 'Rating']], reader)

    # Train model with improved parameters
    svd = SVD(
        n_factors=n_factors,
        biased=True,  # Enable biased learning
        lr_all=0.005,  # Learning rate for all parameters
        reg_all=0.02,  # Regularization for all parameters
        n_epochs=30,   # Number of epochs
        random_state=42
    )

    results = cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=cv_folds, n_jobs=-1)
```

Image 11. Model Training and Validation
(source: writer's archive)

2. Final Model Training

After cross-validation, the final SVD model is trained on the entire dataset to maximize its prediction accuracy. This is the final step in model training, where the entire dataset is used to fit the model and fine-tune its parameters. The image demonstrates the final model training.

```python
# Train final model on full dataset
trainset = data.build_full_trainset()
svd.fit(trainset)
```

Image 12. Final Model Training
(source: writer's archive)

```python
def train_svd_model(df_ratings, n_factors=100, cv_folds=5):
    reader = Reader(rating_scale=(1, 5))
    data = Dataset.load_from_df(df_ratings[['Cust_Id', 'Movie_Id', 'Rating']], reader)

    # Train model with improved parameters
    svd = SVD(
        n_factors=n_factors,
        biased=True,  # Enable biased learning
        lr_all=0.005,  # Learning rate for all parameters
        reg_all=0.02,  # Regularization for all parameters
        n_epochs=30,   # Number of epochs
        random_state=42
    )

    results = cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=cv_folds, n_jobs=-1)

    # Train final model on full dataset
    trainset = data.build_full_trainset()
    svd.fit(trainset)

    return svd, results
```

Image 13. Train SVD Model
(source: writer's archive)

C. Generating Recommendations

To generate movie recommendations for a user, the model predicts ratings for movies that the user has not yet rated. These predictions are then sorted in descending order to present the top recommendations. The image illustrates how recommendations are generated by predicting ratings for unrated movies.

```python
def get_recommendations(svd_model, user_id, df_ratings, df_mov_titles, top_n=20, min_ratings=5):
    # Get movies not rated by user
    user_ratings = df_ratings[df_ratings['Cust_Id'] == user_id]

    # Check if user has minimum number of ratings
    if len(user_ratings) < min_ratings:
        print(f"Warning: User {user_id} has less than {min_ratings} ratings. Recommendations may be less accurate.")

    # Get movies with sufficient ratings
    movie_ratings = df_ratings['Movie_Id'].value_counts()
    popular_movies = movie_ratings[movie_ratings >= min_ratings].index

    # Generate predictions only for movies with sufficient ratings
    predictions = []
    for movie_id in popular_movies:
        if movie_id not in user_ratings['Movie_Id'].values:
            pred = svd_model.predict(user_id, movie_id).est
            predictions.append([movie_id, pred])

    # Create recommendations dataframe
    recs_df = pd.DataFrame(predictions, columns=['Movie_Id', 'Predicted_Rating'])
    recs_df = recs_df.merge(df_mov_titles[['Name', 'Year']],
                            left_on='Movie_Id',
                            right_index=True)

    # Add confidence score based on number of ratings
    movie_rating_counts = df_ratings['Movie_Id'].value_counts()
    recs_df['Number_of_Ratings'] = recs_df['Movie_Id'].map(movie_rating_counts)

    # Sort by predicted rating and number of ratings
    recs_df['Score'] = recs_df['Predicted_Rating'] * np.log1p(recs_df['Number_of_Ratings'])

    return recs_df.sort_values('Score', ascending=False).head(top_n)
```

Image 14. Generating Recommendations
(source: writer's archive)

D. Visualization of RMSE and MAE Performance During Cross-Validation

RMSE (Root Mean Square Error) is used to evaluate the accuracy of the recommender system by measuring the average difference between predicted ratings and actual ratings. Cross-validation is used to assess how well the model generalizes to unseen data. The image shows how the RMSE and MAE performance is visualized during cross-validation to monitor the model's consistency.

```python
def plot_rmse(results):
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

    # Plot RMSE
    ax1.plot(results['test_rmse'], label='RMSE per fold')
    ax1.axhline(y=results['test_rmse'].mean(), color='r', linestyle='--', label='Average RMSE')
    ax1.set_xlabel('Fold')
    ax1.set_ylabel('RMSE')
    ax1.set_title('RMSE per Fold')
    ax1.legend()

    # Plot MAE
    ax2.plot(results['test_mae'], label='MAE per fold')
    ax2.axhline(y=results['test_mae'].mean(), color='r', linestyle='--', label='Average MAE')
    ax2.set_xlabel('Fold')
    ax2.set_ylabel('MAE')
    ax2.set_title('MAE per Fold')
    ax2.legend()

    plt.tight_layout()
    plt.show()
```

Image 15. RMSE and MAE Performance Visualization
(source: writer's archive)

E. Saving Recommendations to a CSV File

Once recommendations are generated, they can be saved to a CSV file for further analysis or reporting. This functionality is important for integrating the recommendation system into larger systems or generating reports. The image shows how the recommendations are saved to a CSV file.

```python
def save_recommendations_to_csv(recommendations, user_id):
    recommendations.to_csv(f'recommendations_for_user_{user_id}.csv', index=False)
    print(f"Recommendations saved to 'recommendations_for_user_{user_id}.csv'.")
```

Image 16. Saving Recommendations to CSV
(source: writer's archive)

F. Usage: Model Training, Evaluation, and Recommendations

Below is the main script that puts all the functions together. This includes loading the data, training the model, visualizing the RMSE performance, generating recommendations, and saving them to a CSV file.

```python
# Usage
if __name__ == "__main__":
    # Load larger sample of data for better model training
    df_ratings, df_mov_titles = load_and_process_data('data', sample_size=2000000)
    print(f"Working with {len(df_ratings)} ratings")

    # Train improved model
    svd_model, results = train_svd_model(df_ratings)
    print("\nCross-validation results:")
    print(f"Average RMSE: {results['test_rmse'].mean():.4f}")
    print(f"Average MAE: {results['test_mae'].mean():.4f}")

    # Visualize model performance
    plot_rmse(results)

    # Get recommendations for a user
    user_ids = [1765963, 1462327]
    for user_id in user_ids:
        recommendations = get_recommendations(svd_model, user_id, df_ratings, df_mov_titles)
        print(f"\nRecommendations for user {user_id}:")
        print(recommendations[['Name', 'Year', 'Predicted_Rating', 'Number_of_Ratings']])
        save_recommendations_to_csv(recommendations, user_id)
```

Image 17. Usage
(source: writer's archive)

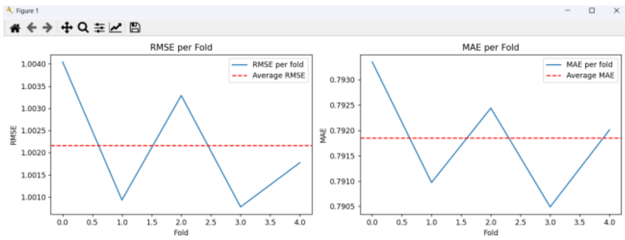


Image 18. Chart RMSE and MAE Performance
(source: writer's archive)

```
Working with 2000000 ratings

Cross-validation results:
Average RMSE: 1.0022
Average MAE: 0.7919
```

Image 19. Final Result 1
(source: writer's archive)

```
Recommendations for user 1765963:
                                               Name    Year  Predicted_Rating  Number_of_Ratings
5                                           Shrek 2  2004.0          5.000000              12749
6                                   The Sixth Sense  1999.0          5.000000              12500
0   Pirates of the Caribbean: The Curse of the Bla...  2003.0          4.874682              15912
7       Lord of the Rings: The Fellowship of the Ring  2001.0          5.000000              12454
9                          Finding Nemo (Widescreen)  2003.0          5.000000              11810
11                                       Braveheart  1995.0          5.000000              11221
16                                      Man on Fire  2004.0          5.000000              11059
17                         The Silence of the Lambs  1991.0          5.000000              10892
24                                Napoleon Dynamite  2004.0          5.000000               9586
30                                              Ray  2004.0          5.000000               9011
38                                     Garden State  2004.0          5.000000               8299
39                                 Secondhand Lions  2003.0          5.000000               8279
31                                Finding Neverland  2004.0          4.945083               8949
41                                 X2: X-Men United  2003.0          4.957918               8093
47                                   Reservoir Dogs  1992.0          5.000000               7466
49                                       Liar Liar  1997.0          5.000000               7437
1                                    Bruce Almighty  2003.0          4.691841              13163
54                                              Elf  2003.0          4.994568               6865
8                                    50 First Dates  2004.0          4.691780              12033
35                                            Speed  1994.0          4.868629               8459
Recommendations saved to 'recommendations_for_user_1765963.csv'.
Warning: User 1462327 has less than 5 ratings. Recommendations may be less accurate.

Recommendations for user 1462327:
                                               Name    Year  Predicted_Rating  Number_of_Ratings
7       Lord of the Rings: The Fellowship of the Ring  2001.0          4.454011              12454
9                          Finding Nemo (Widescreen)  2003.0          4.387946              11810
6                                   The Sixth Sense  1999.0          4.316986              12500
11                                       Braveheart  1995.0          4.355495              11221
17                         The Silence of the Lambs  1991.0          4.348172              10892
0   Pirates of the Caribbean: The Curse of the Bla...  2003.0          4.097031              15912
5                                           Shrek 2  2004.0          4.173452              12749
30                                              Ray  2004.0          4.261833               9011
76                                    The Godfather  1974.0          4.436234               5969
4                                   American Beauty  1999.0          4.028170              12820
40                                 Secondhand Lions  2003.0          4.188818               8279
16                                      Man on Fire  2004.0          4.041661              11059
27                                  A Beautiful Mind  2001.0          4.091455               9116
10                                  The Last Samurai  2003.0          3.991144              11440
69             The Wizard of Oz: Collector's Edition  1939.0          4.233804               6205
63                               When Harry Met Sally  1989.0          4.170270               6615
26                                    Lethal Weapon  1987.0          3.999844               9315
12                              The Bourne Supremacy  2004.0          3.917471              11213
20                             Something's Gotta Give  2003.0          3.938051               9733
8                                    50 First Dates  2004.0          3.844156              12033
Recommendations saved to 'recommendations_for_user_1462327.csv'.
```

Image 20. Final Result 2
(source: writer's archive)

```
Movie_Id,Predicted_Rating,Name,Year,Number_of_Ratings,Score
2452,4.4540110495456116,Lord of the Rings: The Fellowship of the Ring,2001.0,12454,42.000778258819985
3962,4.387946349707656,Finding Nemo (Widescreen),2003.0,11810,41.144836443995544
4306,4.316985788823031,The Sixth Sense,1999.0,12500,40.72456138098464
2782,4.355495472000281,Braveheart,1995.0,11221,40.61774540728577
2862,4.348171528342991,The Silence of the Lambs,1991.0,10892,40.420061876187106
1905,4.097031381714678,Pirates of the Caribbean: The Curse of the Black Pearl,2003.0,15912,39.63833476295887
3938,4.1734516080162924,Shrek 2,2004.0,12749,39.45283395558432
886,4.261833271425628,Ray,2004.0,9011,38.809584746747376
3290,4.436234369179953,The Godfather,1974.0,5969,38.570849510883384
571,4.02817031801853,American Beauty,1999.0,12820,38.101817445506775
1110,4.188818366251477,Secondhand Lions,2003.0,8279,37.78983643156457
1220,4.041660646670465,Man on Fire,2004.0,11059,37.632267142372065
1180,4.091455300910388,A Beautiful Mind,2001.0,9116,37.30546425615061
3624,3.991143625495888,The Last Samurai,2003.0,11440,37.29707224106403
3605,4.233803514203624,The Wizard of Oz: Collector's Edition,1939.0,6205,36.97495702789181
2660,4.1702703317012695,When Harry Met Sally,1989.0,6615,36.686894981595366
1798,3.9998436952342296,Lethal Weapon,1987.0,9315,36.5565259783872
2372,3.917471245230745,The Bourne Supremacy,2004.0,11213,36.53009921303471
30,3.938051053564008,Something's Gotta Give,2003.0,9733,36.16462003405757
1962,3.8441561565042353,50 First Dates,2004.0,12033,36.11773555428953
```

Image 21. Recommendation For User 1462327
(source: writer's archive)

```
Movie_Id,Predicted_Rating,Name
1,3.59944,Dinosaur Planet
2998,3.59944,Lucia Di Lammermoor: Donizetti: Australian Opera
3004,3.59944,Escanaba in da Moonlight
3003,3.59944,The Three Stooges: Merry Mavericks
3002,3.59944,Knightriders
3001,3.59944,Last of the Mississippi Jukes
3000,3.59944,The January Man
2999,3.59944,Bad Bizness
2997,3.59944,The Court-Martial of Billy Mitchell
3006,3.59944,You Got Served: Take it to the Streets
2996,3.59944,Ex-Driver
2995,3.59944,Things I Left in Havana
2994,3.59944,Road to Utopia
2993,3.59944,Wasted
2992,3.59944,The Rundown
2991,3.59944,Motorcycle Mania 3: Jesse James Rides Again
3005,3.59944,As Time Goes By: Series 1 and 2
3007,3.59944,Close Encounters: Proof of Alien Contact
2814,3.59944,Interview with the Assassin
3016,3.59944,Judas Priest: Electric Eye
```

Image 22. Recommendation For User 1765963
(source: writer's archive)

IV. CONCLUSION

The implementation and analysis of SVD-based recommendation systems in Netflix's platform demonstrate several key findings. First, the evolution from traditional SVD to SVD++ has significantly improved the ability to handle sparse data and incorporate implicit feedback, addressing fundamental challenges in large-scale recommendation systems. Integrating explicit ratings and implicit user interactions has proven crucial in generating more accurate and personalized recommendations.

Our experimental results with the Netflix Prize dataset validate the effectiveness of SVD++ in practical applications. The implementation showed consistent performance across different user segments, as evidenced by the RMSE and MAE metrics visualized in our analysis. The stratified sampling approach maintained the data's representativeness while making the computational process more manageable.

The study also highlights the importance of proper data preprocessing and model optimization in building effective recommendation systems. The cross-validation results demonstrate the model's stability and reliability in predicting user preferences. In contrast, the final recommendations generated for specific users (such as users 1462327 and 1765963) show practical applicability in real-world scenarios.

Furthermore, the success of SVD-based approaches in Netflix's recommendation system underscores the broader potential of matrix factorization techniques in solving complex personalization challenges across various digital platforms. The methodology presented in this paper, from data handling to model implementation and evaluation, provides a framework that can be adapted for similar applications in other domains requiring personalized content delivery.

Future research directions could explore the integration of additional contextual factors, the impact of temporal dynamics on user preferences, and the potential of hybrid approaches combining SVD++ with other advanced machine learning techniques to further enhance recommendation accuracy and user experience.

V. APPENDIX

- Github Repository for this paper: https://github.com/varel183/Behind-Netflix-s-Recommendations-The-Influence-of-Singular-Value-Decomposition--SVD-
- YouTube video: https://youtu.be/c6PH7cX6ouY

VI. ACKNOWLEDGMENT

Before everything, the researcher would like to thank the Lord for His Grace and kindness. The researcher would also like to express the biggest gratitude to Dr.

Rila Mandala, Dr. Rinaldi Munir, Dr. Judhi Santoso, and Arrival Dwi Sentosa, M.T for their valuable guidance and teachings throughout the course, which have been essential to the writing of this paper.

As the author of this paper, I would also like to express my sincere gratitude to everyone who has provided support and inspiration throughout the writing process, enabling me to complete the paper titled "Behind Netflix's Recommendations: The Influence of Singular Value Decomposition (SVD)" successfully. I would like to thank my parents for their constant moral support, prayers, and positive encouragement, which have been instrumental in completing this task.

Additionally, I would like to acknowledge the authors whose works laid the foundation for this paper, as well as the journals and articles that provided invaluable insights and contributed significantly to my understanding of the topic.

This paper would not have been possible without the invaluable contributions of the individuals mentioned above. All the assistance and support I have received have been a driving force behind the successful completion of this paper. Thank you very much for all the help and support. I hope this paper can be beneficial to others.

REFERENCES

[1] R. Kumar, B. K. Verma, and S. S. Rastogi, "Social Popularity based SVD++ Recommender System," *International Journal of Computer Applications*, vol. 87, no. 14, February 2014.

[2] A. Nath, A. Ghosh, and A. Mitra, "Building a Movie Recommendation System using SVD algorithm," *International Journal of Computer Sciences and Engineering*, vol. 6, issue 11, November 2018.

[3] S. Wang, G. Sun, and Y. Li, "SVD++ Recommendation Algorithm Based on Backtracking," *College of Electronic Information and Optical Engineering, Nankai University*, July 2020.

[4] M. Rahul et al., "Movie Recommender System using Single Value Decomposition and K-means Clustering," *IOP Conference Series: Materials Science and Engineering*, vol. 1022, 2021.

[5] A. Bhardwaj, C. R. Reddy, and P. Arora, "Movie Recommendation System Using SVD (Letterboxd)," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 12, issue 10, October 2023.

[6] A. Tam, "Using Singular Value Decomposition to Build a Recommender System," *Machine Learning Mastery*, October 2021. [Online]. https://machinelearningmastery.com/using-singular-value-decomposition-to-build-a-recommender-system/ [Accessed 30 December 2024]

[7] J. Tae, "Understanding SVD and Its Application in Recommender Systems," Personal Blog, 2022. [Online]. https://jaketae.github.io/study/svd/ [Accessed 30 December 2024]

[8] "Matrix Factorization Techniques for Recommender Systems," *IEEE Computer*, vol. 42, no. 8, pp. 30-37, 2009. https://www.asc.ohio-state.edu/statistics/dmsl/GrandPrize2009_BPC_BigChaos.pdf [Accessed 30 December 2024]

[9] R. Munir, *Singular Value Decomposition (SVD) (Bagian 1)*, Program Studi Teknik Informatika, STEI-ITB. https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-21-Singular-value-decomposition-Bagian1-2023.pdf [Accessed 30 December 2024]

[10] R. Munir, *Singular Value Decomposition (SVD) (Bagian 2)*, Program Studi Teknik Informatika, STEI-ITB. https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-22-Singular-value-decomposition-Bagian2-2023.pdf [Accessed 30 December 2024]

[11] "What is a Recommendation System?", NVIDIA, 2024. [Online]. https://www.nvidia.com/en-us/glossary/recommendation-system/ [Accessed 30 December 2024]
"Recommendation System dengan Python - Definisi (Part 1)", Medium - Data Folks Indonesia, 2024. [Online]. https://medium.com/data-folks-indonesia/recommendation-system-dengan-python-definisi-part-1-71154dc3f700 [Accessed 30 December 2024]

[12] "Recommender System", Wikipedia. [Online]. https://en.wikipedia.org/wiki/Recommender_system [Accessed 30 December 2024]

[13] "What are Recommender Systems?", GeeksforGeeks, 2024. [Online]. https://www.geeksforgeeks.org/what-are-recommender-systems/ [Accessed 30 December 2024]

[14] "Matrix Factorization (Recommender Systems)", Wikipedia. [Online]. https://en.wikipedia.org/wiki/Matrix_factorization_(recommender_systems) [Accessed 30 December 2024]

[15] "Understanding of Matrix Factorization (MF) and Singular Value Decomposition (SVD)", Medium - Analytics Vidhya, 2024. [Online]. https://medium.com/analytics-vidhya/understanding-of-matrix-factorization-mf-and-singular-value-decomposition-svd-1a38c2d5bbaa [Accessed 30 December 2024]

[16] "Singular Value Decomposition", Wikipedia. [Online]. https://en.wikipedia.org/wiki/Singular_value_decomposition [Accessed 30 December 2024]

[17] "SVD in Recommendation Systems", GeeksforGeeks, 2024. [Online]. https://www.geeksforgeeks.org/svd-in-recommendation-systems/ [Accessed 30 December 2024]

[18] "Matrix Factorization", Surprise Documentation, 2024. [Online]. https://surprise.readthedocs.io/en/stable/matrix_factorization.html [Accessed 31 December 2024]

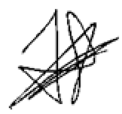
[19] G. Albini, "Building a Movie Recommender Web App from Scratch with SVD and Flask - Part 1", Medium, 2024. [Online]. https://gabri-albini.medium.com/building-a-movie-recommender-web-app-from-scratch-with-svd-and-flask-part-1-ff4d39b837ea [Accessed 31 December 2024]

[20] "Netflix Prize Data", Kaggle, 2024. [Online]. https://www.kaggle.com/datasets/netflix-inc/netflix-prize-data [Accessed 31 December 2024]

STATEMENT

Hereby, I declare that this paper I have written is my own work, not a reproduction or translation of someone else's paper, and not plagiarized.

Bandung, 31 December 2024



Varel Tiara dan 13523008