Evaluating SVD and ANN Algorithms for Movie Recommendations on Netflix

Surya Dharma Putra, *Department of Computer Science, Binus University Jakarta, Indonesia*

Abstract
This study evaluates the effectiveness of two collaborative filtering algorithms Singular Value Decomposition (SVD) and Artificial Neural Network (ANN) in enhancing Netflix's movie recommendation system. The primary focus is to determine which algorithm more effectively harnesses user collaborative behaviors to improve recommendation accuracy. Using the Netflix Prize Dataset, SVD was applied to decompose user-movie interaction matrix, identifying latent features, while ANN was used for dynamic user interactions dan complex model. Each algorithm was measured using MSE, RMSE, and MAE. SVD shows slightly lower rates and higher efficiency with sparse data, while ANN better in dynamic environments where user preference frequently changes. The results suggest SVD provides better speed and robustness, while ANN provides better adaptability. The insights from this study could go beyond entertainment to improve the content delivery system across platforms, increasing user satisfaction and engagement.

Keywords — Movie Recommendation System, Netflix, Artificial Neural Networks, Singular Value Decomposition, Evaluation Metrics

I. Introduction

In an age dominated by digital media, the capability to deliver relevant and personalized content is essential for customer satisfaction and retention. Netflix, a leading streaming service, depends heavily on its recommendation system to suggest movies that align with the diverse tastes of its users. The primary challenge lies in developing and refining these systems to not only improve their accuracy but also adapt effectively to the constantly evolving preferences of users.

Recommender systems are generally classified into three main types: content-based, collaborative, and hybrid recommender systems. Collaborative filtering algorithms are crucial for creating effective recommendation systems.[1] These systems leverage similarities between users and items to tailor recommendations according to individual tastes. Two notable examples of such algorithms are Singular Value Decomposition (SVD) and Artificial Neural Networks (ANN). SVD is renowned for its efficiency in identifying latent factors within large datasets, while ANN, a deep learning algorithm, is valued for its capacity to model complex patterns and adapt dynamically to new information. [2,3,4]

This study aims to evaluate both the SVD and ANN algorithms within Netflix's recommendation system. I will utilize several metrics, including RMSE, MSE, and MAE, to determine which algorithm more effectively harnesses user collaborative behavior to enhance recommendation accuracy.[5] This research will address broad questions regarding the effectiveness of machine learning techniques on a large scale, their real-world applications, and provide insights into the further development of recommendation systems across various sectors.

II. Literature Review

A. Machine Learning

Machine learning encompasses a vast array of algorithms designed to enable computers to make predictions or decisions based on data. This field has been applied across various sectors, from simple prediction tasks to analyzing patterns and behaviors within datasets.

Fig. 1, Machine learning algorithms are generally categorized into supervised, unsupervised, and reinforcement learning. These categories are determined by how the machine interprets its environment and learns from it to achieve specific goals.[6]

- **Supervised Learning:** This involves training a model on a labelled dataset, where the correct output is known, allowing the model to learn over time to predict the output from the input data accurately.[6]
- **Unsupervised Learning:** In contrast, unsupervised learning involves training a model on data that has not been labelled, encouraging the discovery of patterns and relationships within the data.[6]
- **Reinforcement Learning:** This type of learning uses algorithms that learn to react to an environment on their own, maximizing some notion of cumulative reward.[6]

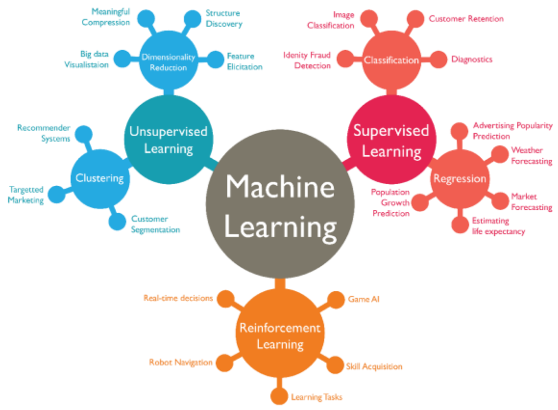


Fig. 1. Machine Learning Categories

Recent advances in deep learning, a subset of machine learning inspired by the structure and function of the human brain, have profoundly impacted AI development. These models have significantly improved capabilities in image and speech recognition, natural language processing, and autonomous vehicle control.[7]

Machine learning's flexibility enhances its utility across various sectors. In healthcare, it predicts patient outcomes, while in finance, it assesses risks and automates trading. Moreover, in recommender systems, ML algorithms analyze user data to tailor content suggestions, crucial for services like streaming platforms. Each sector benefits from its ability to process large datasets and generate valuable insights. [6]

B. Recommender System

Fig. 2, Recommendation systems are crucial for enhancing user experiences by providing personalized recommendations. Utilizing a variety of machine learning algorithms, these systems categorize recommendations based on user preferences. They can be divided into three main categories:

- **Content-Based Filtering:** Recommends items similar to those a user has previously liked, focusing on the attributes of the items themselves. This method analyzes the properties of items and utilizes the user's taste profile to generate recommendations. However, unlike collaborative filtering, it does not take into account the preferences of other users, which can help in identifying broader trends and preferences across a user base. [1]
- **Collaborative Filtering:** This method makes recommendations based on past user behavior and also examines similarities with other users. It is frequently used because it does not require metadata, unlike the content-based method. [1]
- **Hybrid Approaches:** This method combines the two previous methods to improve recommendation performance. [1]

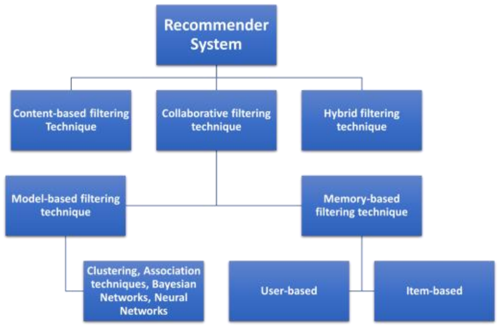

Recommender System
Content-based filtering Technique
Collaborative filtering technique
Hybrid filtering technique
Model-based filtering technique
Memory-based filtering technique
Clustering, Association techniques, Bayesian Networks, Neural Networks
User-based
Item-based

Fig. 2. Recommender System Categories

C. Singular Value Decomposition (SVD)

SVD, a powerful linear algebra technique, is commonly used in recommendation systems to reduce dimensionality by decomposing a matrix into three distinct matrices. [3]This process extracts latent features, such as user preferences and item characteristics, from the data. By revealing the underlying structure of the data, SVD is crucial for generating accurate recommendations.[2]

Mathematical Intuation:

SVD decomposes matrix A into three distinct matrices: U ,Σ and V^T:

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

- **U**: This matrix contains the left singular vectors of A and represents the relationships between users and latent factors.
- **Σ**: This is a diagonal matrix containing the singular values of A. These values measure the amount of information (variance) each latent factor holds. The larger a singular value, the more important the corresponding latent factor.
- **V^T**: The transpose of V contains the right singular vectors of Aand represents the relationships between items and latent factors.

Application in Recommender Systems:

1. Feature Extraction:

Utilizing SVD helps in simplifying the user-item matrix by distilling it down to its most significant features. This reduction helps in mitigating issues like overfitting and sparsity, which are common in large datasets.[3]

2. Incremental SVD:

As discussed by Zhou et al. (2015), the incremental SVD technique updates the decomposed matrices as new data becomes available, such as new ratings from users. This method is vital for maintaining an up-to-date recommendation model that adapts to new interactions without the need for reprocessing the entire data set each time. [3]

The update formula can be conceptualized as adjusting **U**, **Σ**, and **V^T** incrementally when new observations are added to **A**. This ensures the model remains current and reflects the most recent data.[3]

D. *Artificial Neural Networks (ANN)*

Fig. 3, ANN is a computational model inspired by the human brain's network of neurons. It comprises layers of nodes—input, hidden, and output—each connected by adjustable weights that facilitate information processing throughout the network. [8]

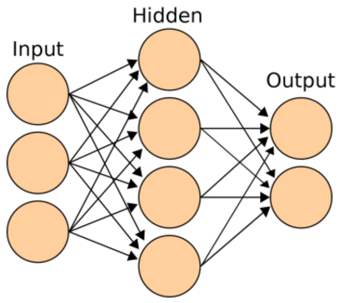


Fig. 3. ANN Illustration

Mathematical Intuition:

ANN operates by learning data representations, with each layer performing specific transformations on incoming inputs. Neurons apply a weighted sum to these inputs and then pass the result through an activation function. This process introduces non-linearity, enabling the network to learn complex patterns. [4]

$$o_i=\phi(\textstyle\sum_j(w_{ij}x_j+b_i))$$

- $\mathbf{O_i}$: output
- $\boldsymbol{\phi}$ **:** activation function
- $\mathbf{w_{ij}}$: weights
- $\mathbf{x_j}$ **:** inputs
- $\mathbf{b_i}$: bias term

Training involves adjusting these weights and biases based on the error between the predicted outputs and the actual data, using algorithms like backpropagation. [8]

Applications in Recommender Systems

Artificial Neural Networks (ANNs) have significantly impacted the field of recommender systems, enhancing the ability to intricately model interactions between users and items. The study by Yücebaş in 2019 on the MovieANN system illustrates how ANNs can be combined with other recommendation techniques to create hybrid systems. These advanced systems utilize the strengths of ANNs to manage vast, dynamic datasets and decipher complex, non-linear relationships that simpler models may not effectively capture. [4]

E. *Evaluation Metric for Recommender System (SVD & ANN)*

In recommender system analysis, models like Singular Value Decomposition (SVD) and Artificial Neural Networks (ANN) are evaluated using various performance metrics. Among the most critical are Root Mean Square Error (RMSE), Mean Squared Error (MSE), and Mean Absolute Error (MAE). These metrics are essential for quantifying the precision of the system's predictions and play a pivotal role in assessing their effectiveness. [5]

Mean Squared Error (MSE) calculates the average of the squared differences between predicted and actual values. It is a risk metric that quantifies the expected value of the square of the deviation, highlighting the average error magnitude in a dataset. [5]

Root Mean Square Error (RMSE) is calculated as the square root of the average of all squared errors. This approach of taking the square root ensures that RMSE values are expressed in the same units as the data, which simplifies the interpretation of the results.

Mean Absolute Error (MAE) computes the average of the absolute differences between the predicted values and actual observations. This metric does not consider the direction of the errors, treating all individual deviations with equal importance.

$$MAE=\frac{1}{N}\sum_{i=1}^{N}|y_i-\hat{y}|$$

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

$$RMSE=\sqrt{MSE}=\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i-\hat{y})^2}$$

$$R^2=1-\frac{\sum(y_i-\hat{y})^2}{\sum(y_i-\bar{y})^2}$$

Where,
$\hat{y}$ – *predicted value of y*
$\bar{y}$ – *mean value of y*

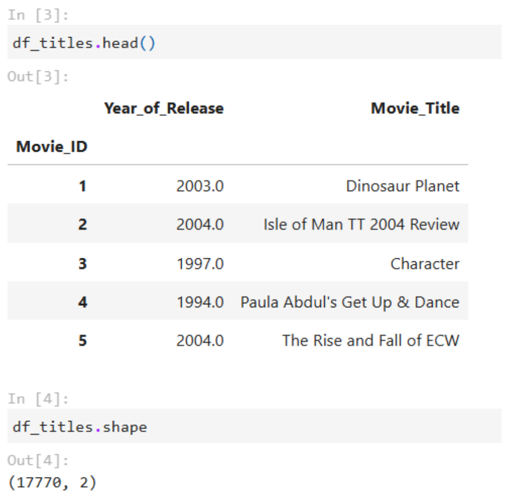
Fig. 4. RMSE, MSE, and MAE Formula

III. METHODOLOGY

A. *Dataset Details*

Dataset taken from Kaggle (Netflix Price Data): https://www.kaggle.com/datasets/netflix-inc/netflix-prize-data

The dataset employed in this study includes user ratings for movies, sourced from Netflix. Fig. 5, 6, It contains ratings for **17,700 different movies**, with a total of **100,480,507 ratings** provided by thousands of users on a scale from 1 to 5. The data is organized into features such as **user_id, movie_id, rating, and date_rating**, distributed across four text files named **combined_data_1 to combined_data_4** stored in txt format. Additionally, **movie_id, movie_titles, Year_of_Release** are listed in a separate dataset stored in CSV format under the name **movie_titles.**

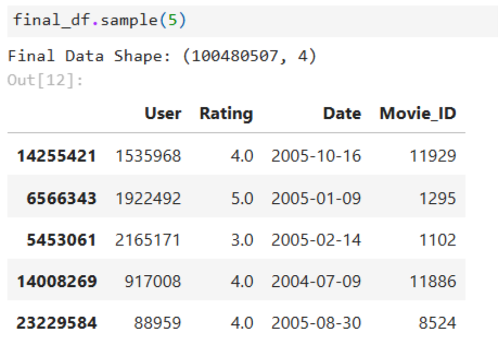


```
In [3]:
df_titles.head()
Out[3]:
```

	Year_of_Release	Movie_Title
Movie_ID		
1	2003.0	Dinosaur Planet
2	2004.0	Isle of Man TT 2004 Review
3	1997.0	Character
4	1994.0	Paula Abdul's Get Up & Dance
5	2004.0	The Rise and Fall of ECW

```
In [4]:
df_titles.shape
Out[4]:
(17770, 2)
```

Fig. 5. Total Movie in Dataset



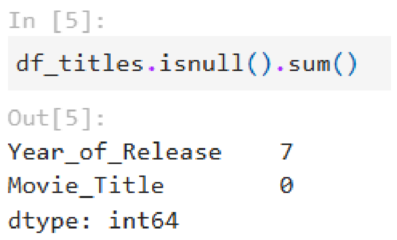
```
final_df.sample(5)
Final Data Shape: (100480507, 4)
Out[12]:
```

	User	Rating	Date	Movie_ID
14255421	1535968	4.0	2005-10-16	11929
6566343	1922492	5.0	2005-01-09	1295
5453061	2165171	3.0	2005-02-14	1102
14008269	917008	4.0	2004-07-09	11886
23229584	88959	4.0	2005-08-30	8524

Fig. 6. Total Rating in Dataset

B. Data Pre-Processing

- **Check Missing Value in movie_titles dataset**



```
In [5]:
df_titles.isnull().sum()
Out[5]:
Year_of_Release    7
Movie_Title        0
dtype: int64
```

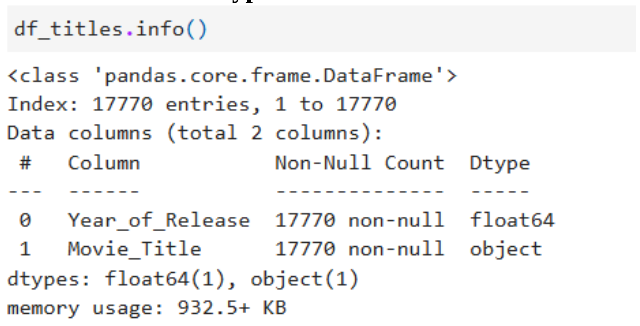
Fig. 7. Missing Value in movie_titles dataset

Fig. 7, There are 7 missing release years in this dataset, I will try to input them manually by searching for the release year of each film.

According to different sources:

- **Ancient Civilizations: Athens and Greece** -> 2001 (https://www.amazon.com/Ancient-Civilizations-Athens-Greece/dp/B00005MKKF)
- **Ancient Civilizations: Rome and Pompeii** -> 2001 (https://www.amazon.com/Ancient-Civilizations-Athens-Greece/dp/B00005MKKF)
- **Ancient Civilizations: Land of the Pharaohs** -> 2001 (https://www.amazon.com/Rome-Pompeii/dp/B081ZNS4J8/)
- **Eros Dance Dhamaka** -> 1998 (https://www.neverdiemedia.com/products/eros-dance-dhamaka)
- **Jimmy Hollywood** -> 1994 (https://www.imdb.com/title/tt0110197/)
- **Hote Hote Pyaar Ho Gaya** -> 1999 (https://www.imdb.com/title/tt0206020/)
- **Roti Kapada Aur Makaan** -> 1974 (https://www.imdb.com/title/tt0072100/)

- **Check Data Types**

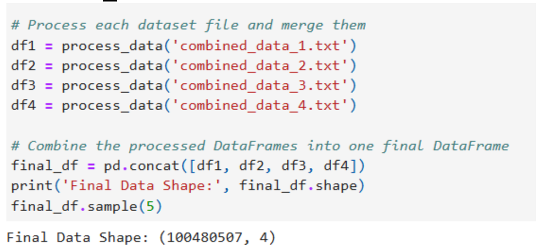


```
df_titles.info()
<class 'pandas.core.frame.DataFrame'>
Index: 17770 entries, 1 to 17770
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Year_of_Release  17770 non-null  float64
 1   Movie_Title      17770 non-null  object
dtypes: float64(1), object(1)
memory usage: 932.5+ KB
```

Fig. 8. movie_titles Data_Types

Fig. 8, For the release year, the data is better in int than float so I typecast it.

- **Combine it with combined_data 1-combined_data 4 dataset**

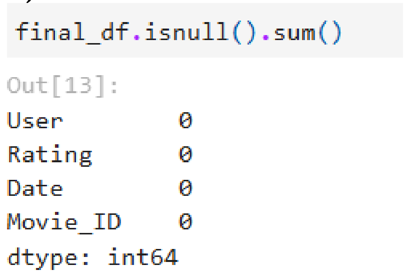


```
# Process each dataset file and merge them
df1 = process_data('combined_data_1.txt')
df2 = process_data('combined_data_2.txt')
df3 = process_data('combined_data_3.txt')
df4 = process_data('combined_data_4.txt')

# Combine the processed DataFrames into one final DataFrame
final_df = pd.concat([df1, df2, df3, df4])
print('Final Data Shape:', final_df.shape)
final_df.sample(5)
Final Data Shape: (100480507, 4)
```

Fig. 9. Combine all 4 datasets

Fig. 9, After being combined the data became 100480507, pretty large data.

- **Check Missing Value in Combine Dataset (after combined)**

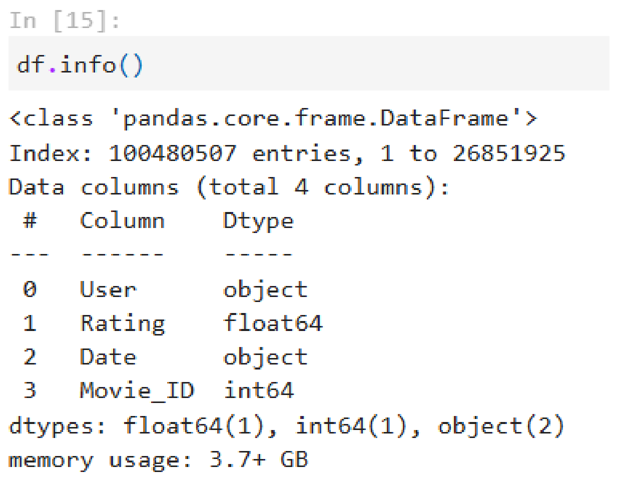


```
final_df.isnull().sum()
Out[13]:
User        0
Rating      0
Date        0
Movie_ID    0
dtype: int64
```

Fig. 10. Missing Value in combine dataset

Fig. 10, There are no missing values.

- **Check Data Types in Combine Dataset**



```
In [15]:
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 100480507 entries, 1 to 26851925
Data columns (total 4 columns):
 #   Column    Dtype  
---  ------    -----  
 0   User      object 
 1   Rating    float64
 2   Date      object 
 3   Movie_ID  int64  
dtypes: float64(1), int64(1), object(2)
memory usage: 3.7+ GB
```

Fig. 11. Combine Data Types

Fig. 11, All variables are in the correct format.

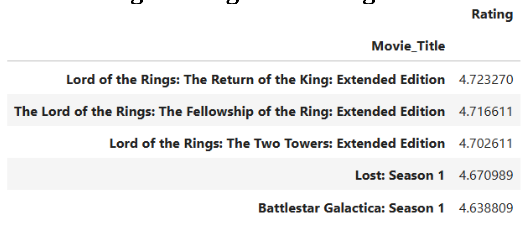
C. Model Implemented

The models utilized in this study are two collaborative filtering algorithms, namely Singular Value Decomposition (SVD) and Artificial Neural Networks (ANN). These models are employed to analyze and predict user ratings based on user_id and movie_id, aiming to enhance movie recommendations on Netflix. SVD is selected for its proficiency in decomposing the user-movie interaction matrix to uncover latent factors. [2,3] In contrast, ANN is chosen for its robust capability to learn complex data patterns and adapt dynamically to new information. [4,8]

IV. Results

A. EDA

- **Movie Average Rating Descending Order**

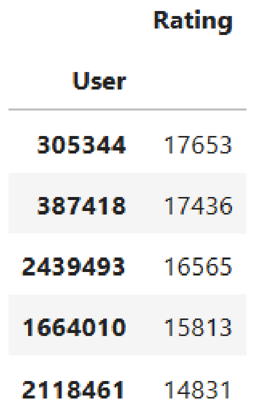


Movie_Title	Rating
Lord of the Rings: The Return of the King: Extended Edition	4.723270
The Lord of the Rings: The Fellowship of the Ring: Extended Edition	4.716611
Lord of the Rings: The Two Towers: Extended Edition	4.702611
Lost: Season 1	4.670989
Battlestar Galactica: Season 1	4.638809

Fig. 12. Movie Average Rating Descending Order

Fig. 12, We can see some of the highest rated films, but often this is biased because it is possible that films that only a few users have rated are also included in the calculation.

- **User with the Most Movie Rated**



User	Rating
305344	17653
387418	17436
2439493	16565
1664010	15813
2118461	14831

Fig. 13. User with the Most Movie Rated

Fig. 13, We can see which User with the most movie rated

- **Distribution of Ratings (Movies and Users)**

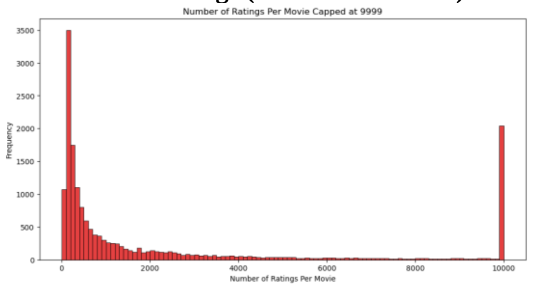


Fig. 14. Distribution of Movies (Capped at 9999)

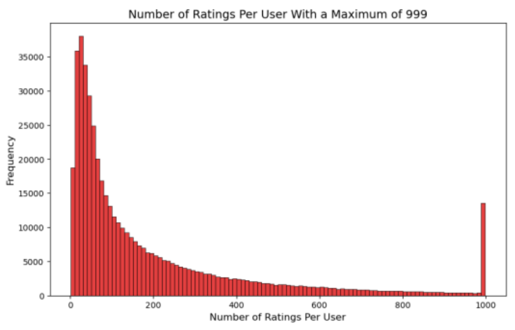


Fig. 15. Distribution of Users Capped at 999)

Fig. 14, 15 , The histograms indicate that most movies and users have a low number of ratings, with a few exceptions that have significantly higher ratings. Given the large dataset size of **100 million ratings**, I decided to focus my recommendation system models on a subset of the data—specifically, users who have rated at least **1000 movies** and movies that have received at least **10,000 ratings**. This filtering will help manage the vast data volume by reducing noise and ensuring the mo
del focuses on more reliable, frequently reviewed content, potentially improving the system's predictive accuracy and relevance.

- **Joint Distribution of Ratings and Number of Ratings**

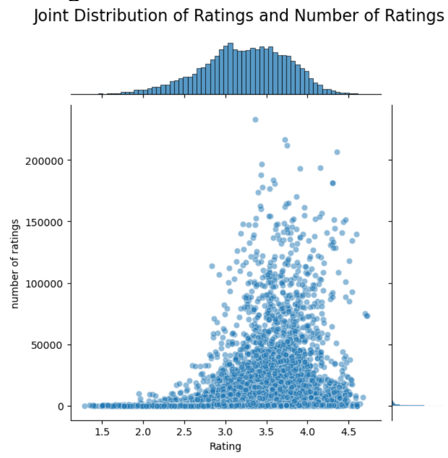


Fig. 16. Joint Distribution of Ratings and Number of Ratings

Fig. 16, The joint distribution plot reveals that the majority of ratings fall between **3 and 4**, suggesting that movies are typically well-received. There's a clear pattern showing that films with **higher ratings** often have **more reviews**, indicating that popular movies tend to receive higher ratings. Additionally, the trend leans towards **higher ratings**, showing that users generally prefer to give positive ratings to movies.

- **Movie Ratings Distribution**

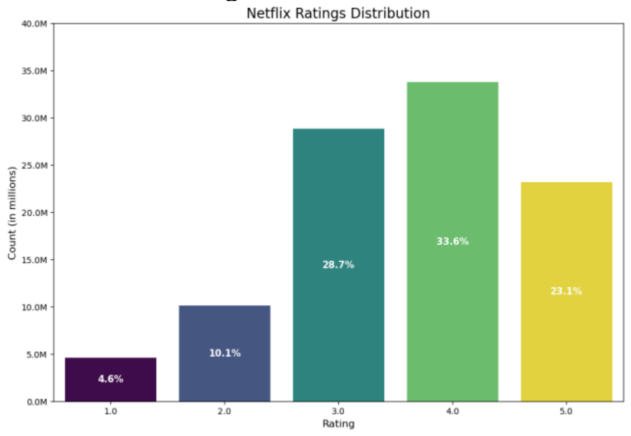


Fig. 17. Netflix Ratings Distribution

Fig. 17, From the distribution above, it can be seen that **most of the ratings given by users are 4 followed by 3** which represents about **62.3%** of the entire distribution. From the distribution, it is also clear that movies on Netflix are **rarely below 3**.

- **Time Series Number of Movies per Year**

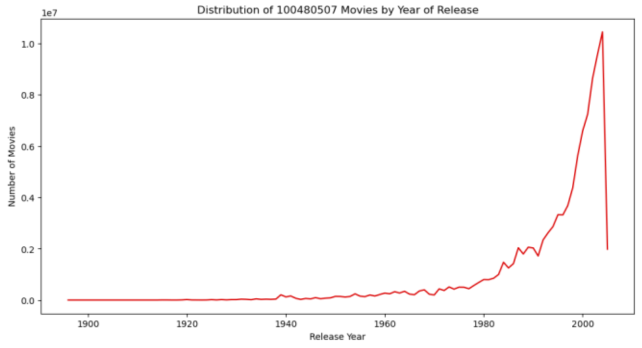


Fig. 18. Time Series Number of Movies per Year

Fig. 18, As the **20th century progressed**, the **number of film releases** increased steadily. The recent decline observed towards the end of the timeline is likely due to **incomplete data entries** for the last year.

B. Model Preparation

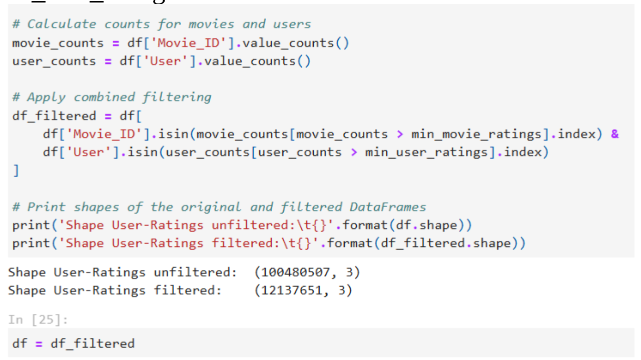
- **Feature Selection**

df **=** df[['User', 'Movie_ID', 'Rating']]

The data that will be input into the model include only the following three attributes: user_id, movie_id, and rating. The date and year of release are excluded because they do not significantly contribute to the predictive accuracy of the model in this context. The focus is on capturing the essential user-movie interactions that directly influence the recommendation output, streamlining the computational process and enhancing model efficiency.

- **Filtering Data to Focus on High-Volume Movies and Users**

min_movie_ratings = 10000
min_user_ratings = 1000



```
# Calculate counts for movies and users
movie_counts = df['Movie_ID'].value_counts()
user_counts = df['User'].value_counts()

# Apply combined filtering
df_filtered = df[
    df['Movie_ID'].isin(movie_counts[movie_counts > min_movie_ratings].index) &
    df['User'].isin(user_counts[user_counts > min_user_ratings].index)
]

# Print shapes of the original and filtered DataFrames
print('Shape User-Ratings unfiltered:\t{}'.format(df.shape))
print('Shape User-Ratings filtered:\t{}'.format(df_filtered.shape))
```

```
Shape User-Ratings unfiltered:  (100480507, 3)
Shape User-Ratings filtered:    (12137651, 3)
```

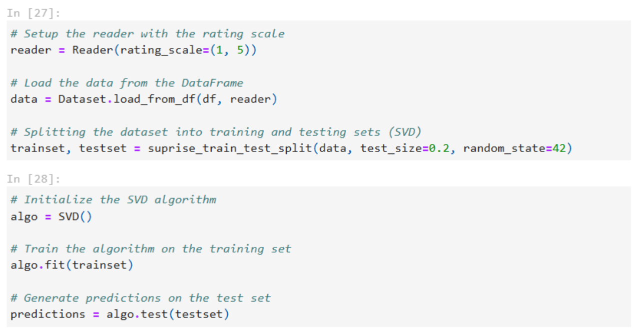
```
In [25]:
df = df_filtered
```

Fig. 19. Filtering Data to Focus on High-Volume Movies and Users

Fig. 19, As discussed before, the dataset will be subsetted to include only movies with a minimum of **10,000 ratings** and users with at least **1,000 ratings**. This strategy not only concentrates on the more dependable data but also cuts down on processing time and memory use, enhancing the efficiency of further analysis.

C. Models

- Collaborative Filtering with SVD

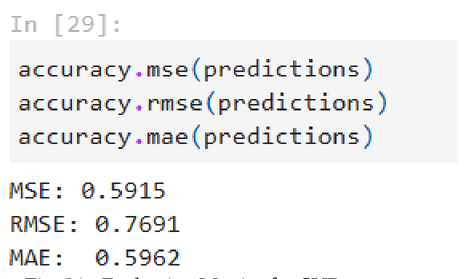


```
In [27]:
# Setup the reader with the rating scale
reader = Reader(rating_scale=(1, 5))

# Load the data from the DataFrame
data = Dataset.load_from_df(df, reader)

# Splitting the dataset into training and testing sets (SVD)
trainset, testset = suprise_train_test_split(data, test_size=0.2, random_state=42)

In [28]:
# Initialize the SVD algorithm
algo = SVD()

# Train the algorithm on the training set
algo.fit(trainset)

# Generate predictions on the test set
predictions = algo.test(testset)
```

Fig. 20. Collaborative Filtering with SVD Code

SVD is selected for its proficiency in decomposing the user-movie interaction matrix to uncover latent factors. Fig.20, Here is the code and its implementation:

1. **Setup the Reader**
   This object is initialized with a specific rating scale, since the rating is between 1 to 5, we set it into that specific rating scale. This informs the model of the rating values that it should expect in the dataset.
2. **Load the Dataset**
   The data load into suitable format for the model using Dataset.load_from_df() function.
3. **Splitting the Dataset:**
   The data is split into train and test set with 80% of data is the trainset and the remaining is the test set for testing, and random_state = 42 is just to make sure the split is reproducible.
4. **Initialize the SVD Algorithm:**
   Run the SVD algorithm for matrix factorization to predict missing entries in the user-movies interaction matrix.
5. **Train the Algorithm:**
   The model is trained on the trainset, the model learns the latent factors that explain the observed rating that helps in predicting unseen ratings
6. **Generate Predictions**
   Make predictions to evaluate how well the model can predict the movie ratings in the test set.
7. **Evaluate the Model**



```
In [29]:
accuracy.mse(predictions)
accuracy.rmse(predictions)
accuracy.mae(predictions)

MSE:  0.5915
RMSE: 0.7691
MAE:  0.5962
```

Fig. 21. Evaluation Metrics for SVD

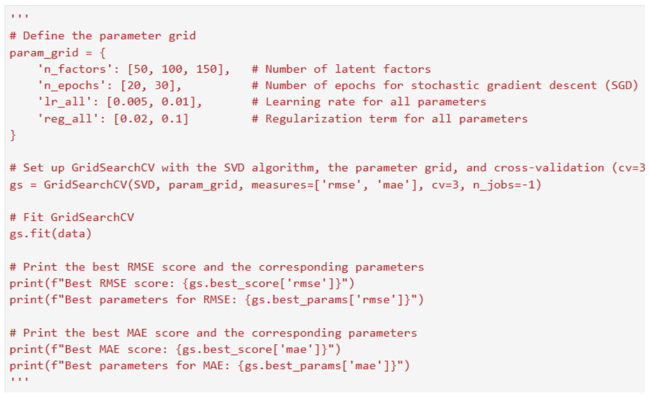
Fig. 21, Evaluation Metrics Overview:

- **MSE (Mean Squared Error)**: Recorded at 0.5915, this metric indicates a moderate level of inaccuracies within the model's predictions. Given the rating scale from 1 to 5, the errors are considered to be within an acceptable range.
- **RMSE (Root Mean Squared Error)**: The model exhibits an average deviation of about 0.7691 points from the actual ratings. This figure serves as a realistic measure of the error magnitude, highlighting the average error in the predictions.
- **MAE (Mean Absolute Error)**: With a value of 0.5962, the MAE suggests that the model's predictions generally do not stray far from the actual values, underscoring the model's precision.

Conclusion:

The SVD model performs competently in estimating movie ratings, with the error metrics indicating **satisfactory but not perfect accuracy**. The proximity of **RMSE to MAE** suggests that the model does not suffer from significant skews from large outliers in its predictions. To enhance the model's performance, one could explore **refining the tuning of its parameters**, **adding richer contextual information**, or **upgrading to more complex algorithms such as SVD++**.

- Hyper tuning SVD parameter



```
'''
# Define the parameter grid
param_grid = {
    'n_factors': [50, 100, 150],   # Number of latent factors
    'n_epochs': [20, 30],          # Number of epochs for stochastic gradient descent (SGD)
    'lr_all': [0.005, 0.01],       # Learning rate for all parameters
    'reg_all': [0.02, 0.1]         # Regularization term for all parameters
}

# Set up GridSearchCV with the SVD algorithm, the parameter grid, and cross-validation (cv=3
gs = GridSearchCV(SVD, param_grid, measures=['rmse', 'mae'], cv=3, n_jobs=-1)

# Fit GridSearchCV
gs.fit(data)

# Print the best RMSE score and the corresponding parameters
print(f"Best RMSE score: {gs.best_score['rmse']}")
print(f"Best parameters for RMSE: {gs.best_params['rmse']}")

# Print the best MAE score and the corresponding parameters
print(f"Best MAE score: {gs.best_score['mae']}")
print(f"Best parameters for MAE: {gs.best_params['mae']}")
'''
```

Fig. 22. Hypertune SVD Code

Parameter:

- **Latent Factors in SVD**: The SVD algorithm utilizes a specified number of latent factors to model the interactions between users and items with varying levels of detail. The parameter grid explores different settings 50, 100, and 150 factors to optimize the model's balance between complexity and predictive accuracy.
- **Training Epochs**: The number of epochs represents how many times the entire dataset is passed through the SVD algorithm during training. Different values such as 20, 30, and 50 epochs are tested to evaluate the model's convergence and performance, aiming to determine the optimal number of iterations for the model to effectively learn from the data.
- **Learning Rate Adjustment**: The learning rate, a key parameter in the training process, controls the speed at which the model updates its knowledge. Different rates are tested to strike a balance where

the model converges quickly without overshooting the optimal solutions.

- **Regularization in Model Training:** Regularization helps in curtailing the model complexity, reducing the risk of overfitting by imposing penalties on excessively large weights. By varying the regularization parameter, the model's ability to generalize to unseen data can be fine-tuned.

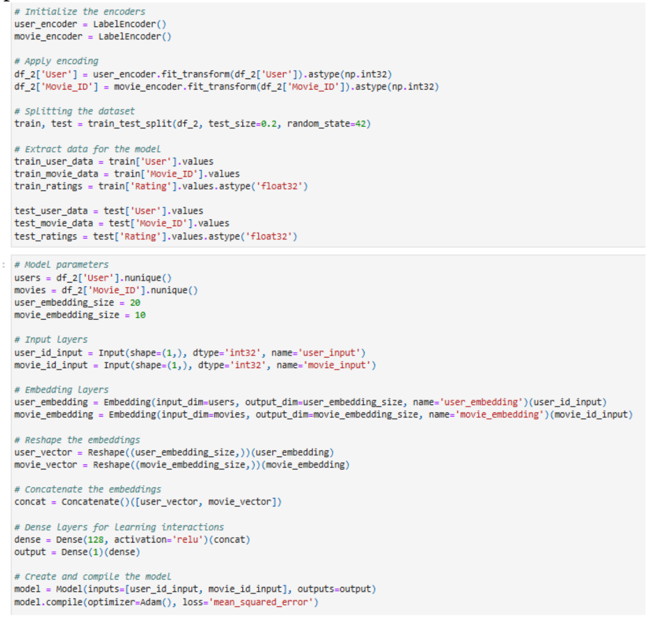
Configuration of Grid Search:

GridSearchCV is configured with the SVD algorithm to systematically test various combinations of parameters across three cross-validation folds. This cross-validation method divides the dataset into three segments, utilizing each segment as a test set while the remainder serves as the training set.

I have tried it, unfortunately even though it has been **4 days** it is still not finished. Due to **time constraints**, I will try it another time

- Collaborative Filtering with ANN (Deep Learning)

As discussed before, ANN is chosen for its robust capability to learn complex data patterns and adapt dynamically to new information. Fig. 22, Here is the code implementation:

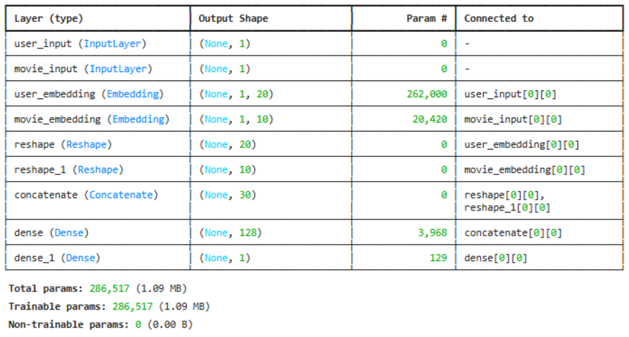


```
# Initialize the encoders
user_encoder = LabelEncoder()
movie_encoder = LabelEncoder()

# Apply encoding
df_2['User'] = user_encoder.fit_transform(df_2['User']).astype(np.int32)
df_2['Movie_ID'] = movie_encoder.fit_transform(df_2['Movie_ID']).astype(np.int32)

# Splitting the dataset
train, test = train_test_split(df_2, test_size=0.2, random_state=42)

# Extract data for the model
train_user_data = train['User'].values
train_movie_data = train['Movie_ID'].values
train_ratings = train['Rating'].values.astype('float32')

test_user_data = test['User'].values
test_movie_data = test['Movie_ID'].values
test_ratings = test['Rating'].values.astype('float32')
```

```
# Model parameters
users = df_2['User'].nunique()
movies = df_2['Movie_ID'].nunique()
user_embedding_size = 20
movie_embedding_size = 10

# Input Layers
user_id_input = Input(shape=(1,), dtype='int32', name='user_input')
movie_id_input = Input(shape=(1,), dtype='int32', name='movie_input')

# Embedding Layers
user_embedding = Embedding(input_dim=users, output_dim=user_embedding_size, name='user_embedding')(user_id_input)
movie_embedding = Embedding(input_dim=movies, output_dim=movie_embedding_size, name='movie_embedding')(movie_id_input)

# Reshape the embeddings
user_vector = Reshape((user_embedding_size,))(user_embedding)
movie_vector = Reshape((movie_embedding_size,))(movie_embedding)

# Concatenate the embeddings
concat = Concatenate()([user_vector, movie_vector])

# Dense Layers for Learning interactions
dense = Dense(128, activation='relu')(concat)
output = Dense(1)(dense)

# Create and compile the model
model = Model(inputs=[user_id_input, movie_id_input], outputs=output)
model.compile(optimizer=Adam(), loss='mean_squared_error')
```

Fig. 22. Collaborative Filtering with ANN (Deep Learning) Code

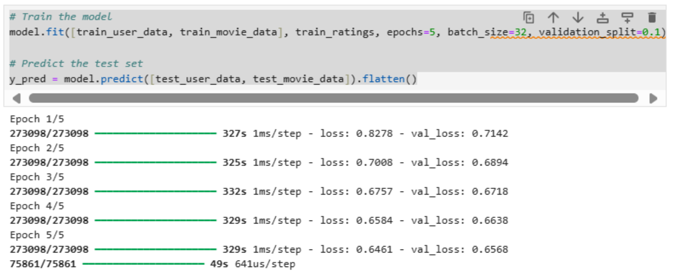
1. **Initialize and Apply Encoders:**
   Change independent variables to suitable numerical format for model input.
2. **Splitting the Dataset:**
   The data is split into train and test set with 80% of data is the trainset and the remaining is the test set for testing, and random_state = 42 is just to make sure the split is reproducible.
3. **Extract Data:**
   Extract user and movie ID from training and testing datasets
4. **Set Model Parameters:**
   Define the number of unique users and movie and sets the size of embeddings for users and movies to capture latent features.
5. **Fig. 23, Define Model Architecture:**
   Input -> user and movie input
   Embedding layers -> users and movies are defined
6. **Add Dense Layers:**
   Add dense layer with activation function "**relu**" to learn the interaction between concatenated embeddings. The final output will predict the rating.
7. **Compile and Fit the Model:**
   The model compiled using "**Adam**" optimizer and **MSE** loss function. After that, the data train and evaluate using test set.



Layer (type)	Output Shape	Param #	Connected to
user_input (InputLayer)	(None, 1)	0	-
movie_input (InputLayer)	(None, 1)	0	-
user_embedding (Embedding)	(None, 1, 20)	262,000	user_input[0][0]
movie_embedding (Embedding)	(None, 1, 10)	20,420	movie_input[0][0]
reshape (Reshape)	(None, 20)	0	user_embedding[0][0]
reshape_1 (Reshape)	(None, 10)	0	movie_embedding[0][0]
concatenate (Concatenate)	(None, 30)	0	reshape[0][0], reshape_1[0][0]
dense (Dense)	(None, 128)	3,968	concatenate[0][0]
dense_1 (Dense)	(None, 1)	129	dense[0][0]

```
Total params: 286,517 (1.09 MB)
Trainable params: 286,517 (1.09 MB)
Non-trainable params: 0 (0.00 B)
```

Fig. 23. Neural Network Model Architecture Summary



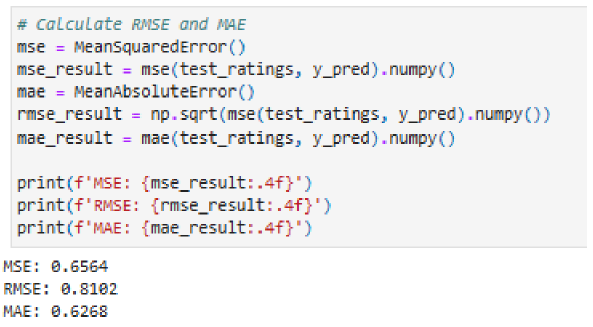
```
# Train the model
model.fit([train_user_data, train_movie_data], train_ratings, epochs=5, batch_size=32, validation_split=0.1)

# Predict the test set
y_pred = model.predict([test_user_data, test_movie_data]).flatten()
```

```
Epoch 1/5
273098/273098 ━━━━━━━━━━━━━━━━━━━━ 327s 1ms/step - loss: 0.8278 - val_loss: 0.7142
Epoch 2/5
273098/273098 ━━━━━━━━━━━━━━━━━━━━ 325s 1ms/step - loss: 0.7008 - val_loss: 0.6894
Epoch 3/5
273098/273098 ━━━━━━━━━━━━━━━━━━━━ 332s 1ms/step - loss: 0.6757 - val_loss: 0.6718
Epoch 4/5
273098/273098 ━━━━━━━━━━━━━━━━━━━━ 329s 1ms/step - loss: 0.6584 - val_loss: 0.6638
Epoch 5/5
273098/273098 ━━━━━━━━━━━━━━━━━━━━ 329s 1ms/step - loss: 0.6461 - val_loss: 0.6568
75861/75861 ━━━━━━━━━━━━━━━━━━━━ 49s 641us/step
```

Fig. 24 Training Process ANN

Fig. 24, The description outlines a machine learning model's training session over five epochs, aimed at predicting movie ratings from user and movie data. The training involves key parameters such as a batch size of 32 and a validation set comprising 10% of the data. As the training progresses through the epoch, both the training loss and the validation loss consistently decrease, showcasing the model's enhanced ability to accurately predict outcomes. This improvement signals that the model is effectively learning from the data. Following the training phase, the model is tested on a separate dataset to assess its predictive performance.

8. Evaluation the Model



```
# Calculate RMSE and MAE
mse = MeanSquaredError()
mse_result = mse(test_ratings, y_pred).numpy()
mae = MeanAbsoluteError()
rmse_result = np.sqrt(mse(test_ratings, y_pred).numpy())
mae_result = mae(test_ratings, y_pred).numpy()

print(f'MSE: {mse_result:.4f}')
print(f'RMSE: {rmse_result:.4f}')
print(f'MAE: {mae_result:.4f}')
```

```
MSE: 0.6564
RMSE: 0.8102
MAE: 0.6268
```

Fig. 25. Evaluation Metrics for ANN

Fig. 25, Evaluation Metrics Overview:

- **MSE (Mean Squared Error):** Recorded at 0.6564, this metric reflects a moderate level of inaccuracies in the model's predictions. It computes the average of the squares of the differences between predicted and actual ratings, indicating that while errors exist, they are within a reasonable range for a rating scale of 1 to 5.
- **RMSE (Root Mean Squared Error):** With a value of 0.8102, this metric shows the standard deviation of the prediction errors. It measures how far the model's predictions deviate from the actual ratings, with an RMSE of approximately 0.8102 suggesting that, on average, the predictions are less than one rating point away from the true values, which offers a practical gauge of the error magnitude.
- **MAE (Mean Absolute Error):** At 0.6268, the MAE indicates that the model predicts ratings with reasonable precision. Typically, the predictions vary from the actual values by just over half a rating point on average, demonstrating the model's general accuracy in forecasting user ratings.

Conclusion:

The ANN model exhibits **solid performance** in predicting movie ratings, although the error metrics reveal there is **potential for improvement**. The **proximity of RMSE to MAE** implies a **consistent error range** across predictions without extreme outliers affecting the overall accuracy significantly. To **optimize the model's effectiveness**, strategies might include more **detailed parameter tuning**, **integrating richer contextual or content-based features**, or **exploring more advanced neural network architectures**.

V. Discussion

A. *SVD vs ANN Results*

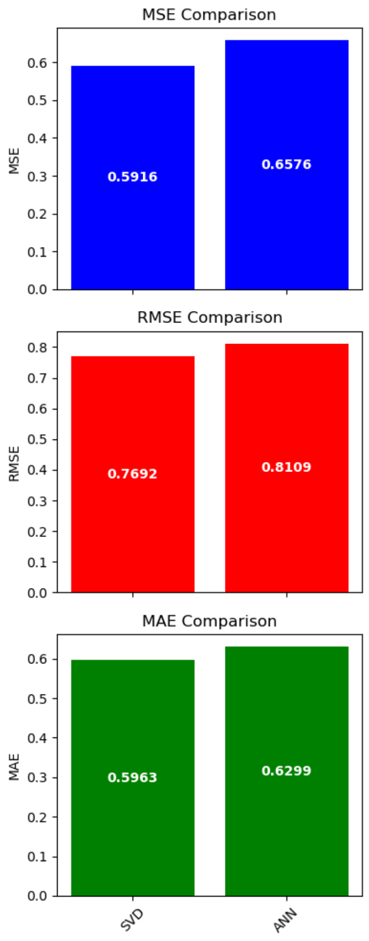


Fig. 26. Model Performance SVD vs ANN

Fig. 26, Model Performance Summary:
The charts compare two distinct models, SVD and ANN, across various standard metrics used to evaluate prediction accuracy.

- **MSE**: SVD demonstrates a more efficient prediction accuracy with a lower MSE of 0.5916, as opposed to ANN's MSE of 0.6576. This suggests that SVD has a tighter grip on minimizing prediction errors.
- **RMSE**: In the realm of RMSE, SVD continues to lead with a score of 0.7692 compared to ANN's 0.8109, indicating smaller deviations in SVD's predictions relative to the true values.
- **MAE**: Consistently, SVD records a lower MAE of 0.5963 against ANN's 0.6299, highlighting its superior ability to predict ratings with minimal error.

Fig. 27, 28, Top 10 Movie Recommendation for User 17

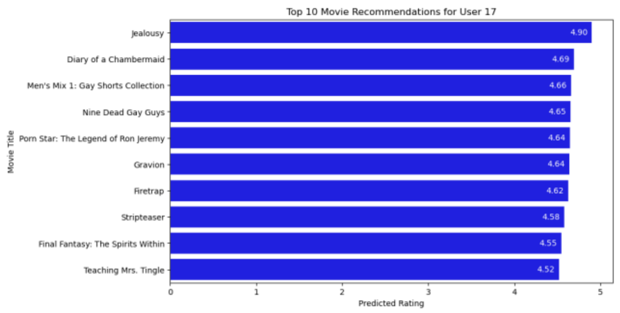


Fig. 27. Top 10 Movie Recommendation for User 17(SVD)

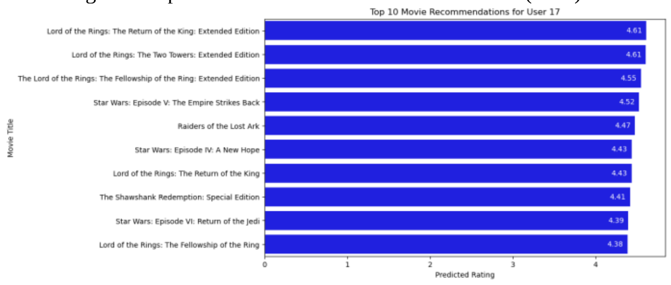


Fig. 28. Top 10 Movie Recommendation for User 17(ANN)

Overview

- **ANN Recommendations**: Features a broad selection of movies, including niche and less-known titles, with predicted ratings reaching up to 4.90.
- **SVD Recommendations**: Focuses on globally recognized and critically celebrated franchises, with top ratings around 4.61.

Conclusion

- **Personalization**: The ANN model excels in delivering personalized movie suggestions that may appeal to specific, individual tastes.
- **Mainstream Appeal**: The SVD model caters effectively to general preferences, highlighting films with widespread acclaim and popularity.

B. *Challenges*

- The original dataset contains 100,480,507 ratings, which is quite large and taxing on computer resources. It has been condensed to 12,137,651 ratings by filtering for users who have rated at least 1,000 times and movies that have received at least 10,000 ratings. Running SVD without hyperparameter tuning on this reduced dataset still takes about 1 hour, and with hyperparameter tuning, it can take up to four days to complete, while ANN takes about 1.5 hours.
- The laptop crashed several times because the RAM required was quite large.

VI. Conclusion

A. *Summary*

The ANN model exhibits **solid performance** in predicting movie ratings, although the error metrics reveal there is **potential for improvement**. The **proximity of RMSE to MAE** implies a **consistent error range** across predictions without extreme outliers affecting the overall accuracy significantly. To **optimize the model's effectiveness**, strategies might include more **detailed parameter tuning**, **integrating richer contextual or content-based features**, or **exploring more advanced neural network architectures**.

B. *Suggestion for Future Works*

- **Integration with Apache Spark and ALS**:
  Consider using Apache Spark's MLib to implement **Alternating Least Squared (ALS).** Spark is more efficient in processing big data like this dataset, which is ideal for handling millions of data like Netflix movie recommendations. ALS is well-suited for collaborative filtering tasks and could potentially improve recommendation accuracy and computational efficiency.
- **Exploration of Ensemble Techniques**
  Investigate the potential of combining SVD and ANN with other machine learning models. By combining multiple algorithms, such as random forests or gradient boosting machines, the individual advantages of each can be utilized to potentially improve the overall effectiveness and robustness of the recommendation system's predictive capabilities.

References

[1] Deepjyoti Roy and Mala Dutta, "A systematic review and research perspective on recommender systems," *Journal of Big Data*, vol. 9, no. 59, 2022, pp. 1–33, doi:10.1186/s40537-022-00592-5.

[2] H. Bhowmick, A. Chatterjee, and J. Sen, "Comprehensive Movie Recommendation System," in Dept. of Data Science, Praxis Business School, Kolkata, India, 2022, pp. 1-8.

[3] X. Zhou, J. He, G. Huang, and Y. Zhang, "SVD-based incremental approaches for recommender systems," *Journal of Computer and System Sciences*, vol. 81, no. 4, pp. 717-733, June 2015.

[4] S. C. Yücebaş, "MovieANN: A Hybrid Approach to Movie Recommender Systems Using Multi Layer Artificial Neural Networks", Çanakkale Onsekiz Mart Üniversitesi Fen Bilimleri Enstitüsü Dergisi, vol. 5, no. 2, pp. 214–232, 2019, doi: 10.28979/comufbed.597093.

[5] B. Rawat and S. K. Dwivedi, "Selecting Appropriate Metrics for Evaluation of Recommender Systems," *I.J. Information Technology and Computer Science*, vol. 11, no. 1, pp. 14-23, Jan. 2019.

[6] E. F. Morales and H. J. Escalante, "A brief introduction to supervised, unsupervised, and reinforcement learning," in *Biosignal Processing and Classification Using Computational Learning and Intelligence*, A. A. Torres-García, C. A. Reyes-García, L. Villaseñor-Pineda, and O. Mendoza-Montoya, Eds. Academic Press, 2022, pp. 111-129. [Online]. Available: https://doi.org/10.1016/B978-0-12-820125-1.00017-8

[7] A. Khodadadi, S. Ghandiparsi, and C.-N. Chuah, "A Natural Language Processing and deep learning based model for automated vehicle diagnostics using free-text customer service reports," *Machine Learning with Applications*, vol. 10, 2022, Art. no. 100424. [Online]. Available: https://doi.org/10.1016/j.mlwa.2022.100424

[8] J. Zupan, "Introduction to Artificial Neural Network (ANN) Methods: What They Are and How to Use Them," *Acta Chimica Slovenica*, vol. 41, no. 3, pp. 327-352, 1994.