INTRODUCTION TO ARTIFICIAL INTELLIGENCE

TOPIC: MOVIE RECOMMENDATIONS SYSTEM

GROUP 3

Team Members

Vu Thuong Tin 20230091 Chu Anh Duc 20230081

Tran Quang Hung 20235502 Phan Dinh Trung 20230093

Instructor:	Assoc. Prof. Le Thanh Huong
Major:	Data Science and Artificial Intelligence
School:	School of Information and Communications Technology

HANOI - 11 - 2024

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION 2

1.1 Problem Statement 2

1.2 Background and Problems of Research 2

1.3 Research Objectives and Conceptual Framework 3

CHAPTER 2. LITERATURE REVIEW 6

2.1 Scope of Research 6

2.2 Related Works 7

CHAPTER 3. BASIC MATHEMATICS 8

3.1 Content - Based Recommendations 8

3.2 Neighborhood - Collaborative Filtering 9

3.3 Matrix Factorization 11

CHAPTER 4. OVERVIEW 13

4.1 Abstract 13

4.2 Data Processing 13

4.3 Model Training - Algorithms 15

CHAPTER 5. NUMERICAL RESULTS 20

5.1 Evaluation Parameters 20

5.2 Result of Evaluating 21

CHAPTER 6. CONCLUSIONS 24

6.1 Summary 24

6.2 Suggestion for Future Works 24

1.1 Problem Statement

Recommendation systems have become an integral part of modern digital platforms, offering personalized suggestions to users based on their preferences and behavior. In the domain of movies, these systems play a crucial role in enhancing user experience by helping individuals navigate the overwhelming number of options available on streaming services, online stores, and other media platform. By leveraging techniques such as collaborative filtering, content-based filtering, and hybrid approaches, movie recommendation systems analyze vast datasets of user interactions, movie metadata, and contextual information to deliver tailored recommendations. The primary objective is to bridge the gap between user preferences and available content, fostering engagement, satisfaction, and retention.

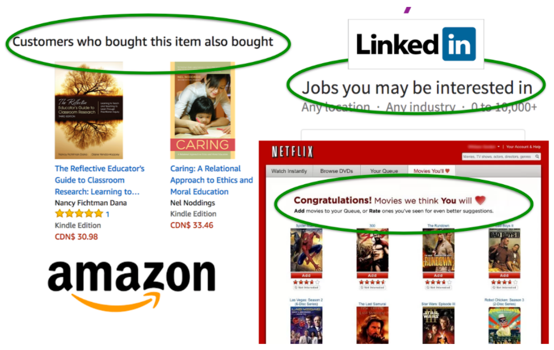

Customers who bought this item also bought
Jobs you may be interested in
Congratulations! Movies we think You will
amazon

Figure 1: Polular Recomendations System

1.2 Background and Problems of Research

Otherwise, recommendation systems is a fairly broad area of Machine Learning and is younger than Classification because the internet has only really exploded in the last 10 - 15 years. There are two main entities in the Recommendation Systems users, and items or users are users. Items are products, such as movies, songs, books, clips, or other users in the friend suggestion problem. The main purpose of Recommender Systems is to predict a user's interest in a certain item, thereby having a suitable recommendation strategy.

There are various challenges faced by Recommendation System. These challenges are Cold Start problem, Data Sparsity, Scalability.

Cold Start Problem: It needs enough users in the system to find a match. For instance, if we want to find similar user or similar item, we match them with the set of available users or items. At initial stage for a new user, his profile is empty as he has not rated any item and the system do not know about his taste, so it becomes difficult for a system to provide him recommendation about any item. Same case can be with new item, as it is not rated by any user because it's new for the user. Both these problem can be resolved by implementing hybrid techniques.

Data Sparsity: The user or rating matrix is very sparse. It is very hard to find users that have rated the same items because most of the user does not rate the items. So it becomes hard to find set of users who rate the items. To give recommendation is really tough when there is less information about any user .

Scalability: Collaborative Filtering use massive amount of data to make reliable better which require more number of resources. As information grows exponentially processing becomes expensive and inaccurate result from this Big data challenge.

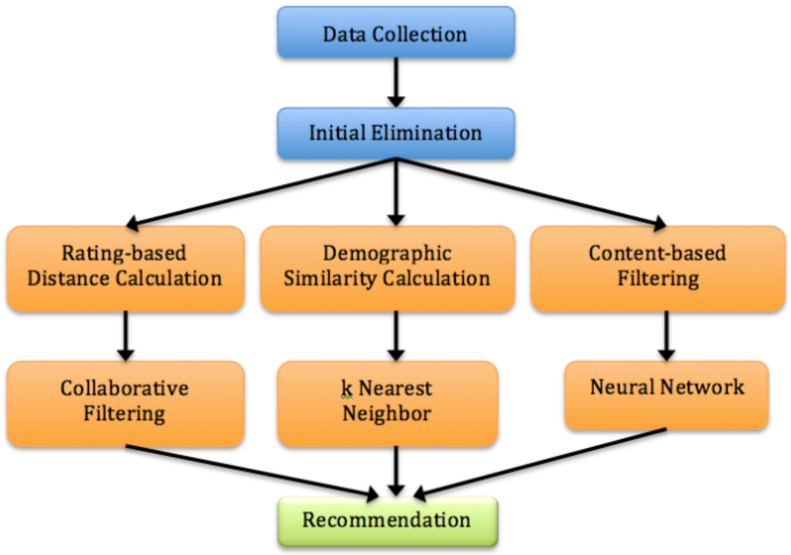
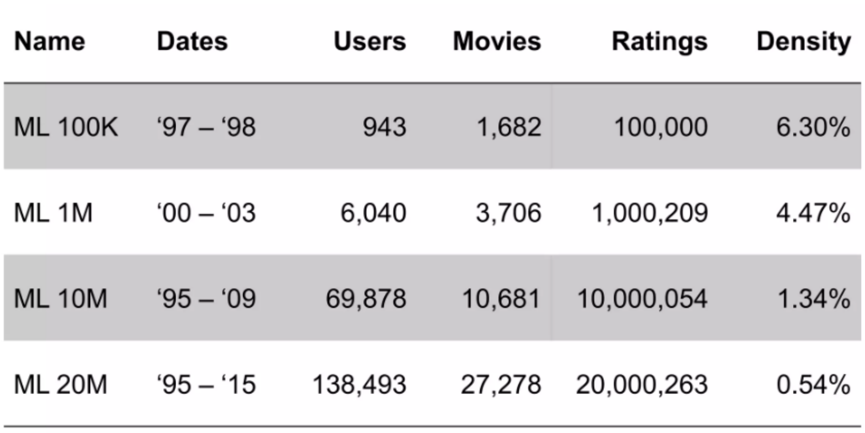


Figure 2: Structure of Recommendations system

1.3 Research Objectives and Conceptual Framework

There are a number of datasets that are available for recommendation research. Among of them, the MovieLens dataset is probably one of the more popular ones. MovieLens is a non-commercial web-based movie recommender system. It is created in 1997 and run by GroupLens, a research lab at the University of Minnesota, in order to gather movie rating data for research purposes. MovieLens data has been critical for several research studies including personalized recommendation and social psychology.



Name	Dates	Users	Movies	Ratings	Density
ML 100K	‘97 – ‘98	943	1,682	100,000	6.30%
ML 1M	‘00 – ‘03	6,040	3,706	1,000,209	4.47%
ML 10M	‘95 – ‘09	69,878	10,681	10,000,054	1.34%
ML 20M	‘95 – ‘15	138,493	27,278	20,000,263	0.54%

Figure 3: Comparision of ML 100k and ML 1M

When working with datasets like ML 100K and ML 1M, there are a few notable challenges to be aware of. First, the large size of ML 1M requires more powerful computational resources and longer processing times, while ML 100K can suffer from overfitting due to small and lack of diverse data. Second, data sparsity reduces the effectiveness of matrix-based algorithms and makes it difficult to predict for less interactive movies or users, especially the cold-start problem when dealing with new users or movies. Third, the uneven quality of the data, with subjective and inconsistent ratings, makes it difficult to distinguish between users who dislike a movie and those who have never watched it. Finally, balancing accuracy and performance is a major challenge, especially when large datasets require more processing complexity but offer the potential to improve recommendation performance. To overcome this, methods such as data size reduction, using association models, and exploiting additional information need to be applied to improve performance and accuracy.

We have proposed the following algorithms Content-Based Recommendations, Neighborhood - Based Collaborative Filtering, Matrix Factorization.

Content - Based Recommendations: The Content-Based Recommendation algorithm focuses on analyzing the characteristics of movies and the user’s interaction history to provide suggestions. This algorithm emphasizes finding movies similar to those the user has liked or watched before by analyzing their content. There are 3 steps.

1. **Analyzing movie features:** Each movie is described by a set of attributes (features), such as genres, director, main actors, release year, Plot descriptions or related keywords. These attributes are represented as content vectors for each movie.

2. **User profile:** A user profile is built based on their watch history, ratings, or interactions. The user profile is represented by aggregating the content vectors of the movies they have rated highly.
3. **Finding similar movies:** After creating the user profile, the algorithm searches for movies with content vectors similar to the user profile vector (usually using similarity measures like cosine similarity). Movies with high similarity scores are recommended.

Neighborhood - Based Collaborative Filtering: Neighborhood-Based Collaborative Filtering is a widely used technique in recommendation systems that relies on the concept of similarity between users or items (movies) to make predictions. Unlike Content-Based Recommendation, which focuses on the attributes of movies, Collaborative Filtering works by analyzing user-item interactions (e.g., ratings) to identify patterns of shared preferences. There are 2 steps.

1. **User-Based Collaborative Filtering:** Focuses on identifying similar users based on their past interactions, *key idea* - Users who have rated or liked similar movies in the past are likely to have similar preferences in the future. Compute similarity between the target user and other users (e.g., using cosine similarity or Pearson correlation on their rating vectors), identify a set of neighboring users. Then, we aggregate the ratings of the neighboring users to predict the target user's rating for a movie.
2. **Item-Based Collaborative Filtering:** Focuses on finding similar movies based on user ratings, *key idea* - Movies that are rated similarly by multiple users are likely to have similar appeal. Compute similarity between movies, for a given movie, find a set of neighboring movies. Then, we predict the user's rating for a movie based on their ratings of similar movies.

Matrix Factorization: Matrix Factorization is a powerful and widely used technique in recommendation systems that leverages linear algebra to predict missing entries in a user-item interaction matrix (e.g., a matrix of user ratings for movies). It is particularly effective for large, sparse datasets and forms the basis of modern recommendation algorithms.

CHAPTER 2. LITERATURE REVIEW

2.1 Scope of Research

The MovieLens 100k dataset is a cornerstone in recommender systems research, providing a rich collection of 100,000 user ratings across 1,682 movies from 943 unique users. Its structured yet compact size offers a practical platform for experimenting with and comparing various recommendation algorithms. This research aims to explore the application of **Content-based recommendations**, **Collaborative filtering**, and **Matrix factorization** methods to maximize accuracy, scalability, and user satisfaction within the MovieLens 100k dataset's context.

Research Objectives

1. **Understand Dataset Properties**

   + Analyze user and movie distributions, rating patterns, and sparsity levels.
   + Evaluate potential biases, such as popularity bias or rating concentration on specific movies or users.

2. **Algorithm Exploration**

   + **Content-based recommendations**: Leverage movie metadata (e.g., genres, titles) to recommend movies similar to those a user has previously rated.

   + **Collaborative filtering**:

   - User - user and item - item methods to identify similarities in rating patterns.
   - Address the cold-start problem by exploring hybrid approaches.

   + **Matrix factorization**: Use techniques like Singular Value Decomposition (SVD) and Alternating Least Squares (ALS) to uncover latent factors representing user and movie characteristics.

3. **Model Implementation and Optimization**

   + Implement selected algorithms using frameworks like Scikit-learn, Surprise, and TensorFlow.

   + Optimize hyperparameters to enhance recommendation quality, addressing overfitting and underfitting.

4. **Evaluation Metrics and Benchmarking**

   + Use metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Precision@K for quantitative performance evaluation.

   + Perform comparative analysis to assess trade-offs between computational efficiency and recommendation accuracy.

5. **Practical Deployment Considerations**

   + Evaluate the feasibility of real-world deployment by analyzing algorithm scalability and responsiveness to dynamic datasets.

   + Explore opportunities for integrating contextual information to improve personalization.

2.2 Related Works

Recommendation systems have become a cornerstone in personalized content delivery, with applications spanning various domains such as e-commerce, online streaming platforms, and social networks. In this section, we review existing applications, focusing on Netflix's recommendation system as a benchmark due to its prominence and sophistication. We also discuss other relevant studies in recommendation algorithms to contextualize the research scope.

1. **Netflix Recommendation System** Netflix's recommendation system is a leading example of how advanced algorithms can drive user engagement and business success. Its multi-layered architecture integrates various recommendation approaches

2. **Other Applications and Studies** Several other platforms and studies contribute valuable insights into recommendation system research

   + **Amazon and E-Commerce Recommendations**: Amazon's system heavily relies on item-item collaborative filtering, prioritizing scalability for its vast catalog. The use of implicit feedback and co-purchase data has proven effective in generating real-time suggestions.

   + **YouTube and Video Recommendations**: YouTube employs a two-stage architecture: candidate generation using collaborative filtering and ranking based on deep learning models. The system factors in watch time, clicks, and engagement metrics to improve relevance.

   + **Spotify and Music Recommendations**: Spotify integrates collaborative filtering, audio signal processing, and natural language processing to create personalized playlists and suggestions. Their system exemplifies how domain-specific features (e.g., acoustic properties) can enhance user satisfaction.

3.1 Content - Based Recommendations

1. **Ridge regression**

   Loss function

   The loss function of the Ridge regression is defined as:

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \mathbf{x}_i^\top \mathbf{w}\right)^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$

   Explanation

   Least squares error

$$\frac{1}{2N}\sum_{i=1}^{N}\left(y_i - \mathbf{x}_i^\top \mathbf{w}\right)^2$$

   + This is the sum of squared errors between the actual value y_i and the predicted value $\mathbf{x}_i^\top \mathbf{w}$.

   + This term measures the accuracy of the model.

   ℓ_2 **Regularization term**

$$\frac{\lambda}{2}\|\mathbf{w}\|_2^2 = \frac{\lambda}{2}\sum_{j=1}^{d} w_j^2$$

   + This term penalizes large values of the weights w, helping to reduce overfitting.

   + Large λ: prioritizes reducing model complexity.

   + Small λ: focuses on minimizing prediction error.

   Objective The goal is to optimize w to minimize the loss function $\mathcal{L}(\mathbf{w})$:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}}\, \mathcal{L}(\mathbf{w})$$

   Closed-Form solution: Given the feature matrix $\mathbf{X}$ ($N \times d$) and the target vector $\mathbf{y}$ ($N \times 1$), the solution to Ridge Regression is:

$$\mathbf{w}^* = \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}\right)^{-1} \mathbf{X}^\top \mathbf{y}$$

   where:

+ $\mathbf{I}$: identity matrix.

+ λ: regularization coefficient.

2. **Utility matrix**

   Formally, a utility matrix $\mathbf{Y}$ is a matrix of size $M \times N$, where:

   + M: Number of items.

   + N: Number of users.

   + y_{mn}: The utility (e.g., rating, score, or preference) assigned by user n to item m.

   Key keatures of a utility matrix

   (a) **Sparse representation**

   In most real-world applications, the utility matrix is sparse because:

   + Not all users rate or interact with all items.

   + Many entries y_{mn} are unknown or missing.

   (b) **Numerical entries**

   The entries in the matrix are typically numerical, such as:

   + Ratings (e.g., 1–5 stars in a movie-rating system).

   + Binary values (e.g., 0 for no interaction, 1 for interaction).

   + Implicit scores (e.g., frequency of interaction or purchase history).

   (c) **Approximation:**

   The goal of many machine learning or optimization methods (e.g., matrix factorization) is to approximate the missing values in the utility matrix by learning patterns from the observed entries.

3.2 Neighborhood - Collaborative Filtering

1. **Similarity function** Is an important concept in machine learning, data processing, and problems involving the comparison of two objects. This function measures the degree of similarity between two objects (e.g., vectors, data points, or sets) based on specific criteria.

2. **General formula** A similarity function $\text{sim}(\mathbf{a}, \mathbf{b})$ takes two objects $\mathbf{a}$ and $\mathbf{b}$ as input and returns a value within a defined range, typically:

$$\text{sim}(\mathbf{a}, \mathbf{b}) \in [0, 1] \quad \text{or} \quad \text{sim}(\mathbf{a}, \mathbf{b}) \in [-1, 1]$$

Cosine similarity: Measures the angle between two vectors in a feature space. It is calculated by:

$$\text{sim}_{\text{cosine}}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \|\mathbf{b}\|}$$

The value ranges from $[-1, 1]$.

3. **Prediction rating**

Formula

The general formula for the predicted rating $\hat{r}_{ui}$ in a recommendation system is:

$$\hat{r}_{ui} = f(u, i)$$

Where:

+ $\hat{r}_{ui}$ is the predicted rating for user u on item i.

+ $f(u, i)$ is a function that calculates the predicted rating based on user u's preferences and item i's characteristics.

In the context of different methods, the formula can vary :

User-User Collaborative filtering

$$\hat{r}_{ui} = \frac{\sum_{v \in N(u)} \text{sim}(u, v) \cdot r_{vi}}{\sum_{v \in N(u)} |\text{sim}(u, v)|}$$

Where:

+ $N(u)$ is the set of users most similar to user u.

+ sim(u, v) is the similarity between the user u and the user v.

+ r_{vi} is the rating that user v gave to item i.

Item-Item Collaborative filtering

$$\hat{r}_{ui} = \frac{\sum_{j \in N(i)} \text{sim}(i, j) \cdot r_{uj}}{\sum_{j \in N(i)} |\text{sim}(i, j)|}$$

Where:

+ $N(i)$ is the set of items most similar to item i.

+ sim(i, j) is the similarity between item i and item j.

+ r_{uj} is the rating that user u gave to item j.

3.3 Matrix Factorization

1. **Gradient's Descent:** Given the loss function for matrix factorization, the updates for the user and item feature matrices P and Q using gradient descent are as follows:

   For the User feature Matrix P

   The gradient of the loss function with respect to p_{ik} is

   $$\frac{\partial \text{Loss}}{\partial p_{ik}} = -2\sum_{j\in\kappa_i}(r_{ij} - p_i^T q_j)q_{jk} + 2\lambda p_{ik}$$

   Where:

   + κ_i is the set of items rated by user i
   + r_{ij} is the actual rating for user i and item j
   + $p_i^T q_j$ is the predicted rating for user i and item j
   + λ is the regularization parameter
   + q_{jk} is the feature of item j corresponding to the latent factor k

   The update rule for p_{ik} is:

   $$p_{ik} := p_{ik} - \eta\frac{\partial \text{Loss}}{\partial p_{ik}}$$

   where η is the learning rate.

   For the Item feature Matrix Q

   The gradient of the loss function with respect to q_{jk} is:

   $$\frac{\partial \text{Loss}}{\partial q_{jk}} = -2\sum_{i\in\kappa_j}(r_{ij} - p_i^T q_j)p_{ik} + 2\lambda q_{jk}$$

   Where:

   + κ_j is the set of users who rated item j
   + $p_i^T q_j$ is the predicted rating for user i and item j
   + p_{ik} is the feature of user i corresponding to the latent factor k
   + λ is the regularization parameter

   The update rule for q_{jk} is:

$$q_{jk} := q_{jk} - \eta \frac{\partial \text{Loss}}{\partial q_{jk}}$$

where η is the learning rate.

2. **Singular Value Decomposition - SVD**

$$\hat{r}_{ui} = \mathbf{p}_u^T \mathbf{q}_i$$

Where:

+ $\mathbf{p}_u$ is the latent feature vector for the user u

+ $\mathbf{q}_i$ is the latent feature vector for item i

+ $\mathbf{p}_u^T \mathbf{q}_i$ is the dot product between the vectors of features of the user and the item

CHAPTER 4. OVERVIEW

4.1 Abstract

This study outlines a comprehensive methodology pipeline for building and deploying a recommendation system using the MovieLens 100k dataset. The pipeline is divided into four major stages: Data Processing, Model Training, Deployment. The recommendation algorithms used in this research are Content-based Filtering, Collaborative Filtering, and Matrix Factorization. Each stage plays a crucial role in ensuring the effectiveness, accuracy, and scalability of the recommendation system. Below is a detailed description of each step in the methodology.

4.2 Data Processing

We will use dataset Movielen100k: `https://files.grouplens.org/datasets/movielens/ml-100k.zip`

1. **Collection Data**

   Here are brief descriptions of the data.

   - **`u.data`** – The full u data set, 100000 ratings by 943 users on 1682 items. Each user has rated at least 20 movies. Users and items are numbered consecutively from 1. The data is randomly ordered. This is a tab-separated list of:

     ```
     user id|item id|rating|timestamp
     ```

   - **`u.info`** – The number of users, items, and ratings in the u data set.
   - **`u.item`** – Information about the items (movies); this is a tab-separated list of:

     ```
     movie id|movie title|release date|video release date|
     IMDb URL|unknown|Action|Adventure|Animation|
     Children's|Comedy|Crime|Documentary|Drama|Fantasy|
     Film-Noir|Horror|Musical|Mystery|Romance|Sci-Fi|
     Thriller|War|Western
     ```

     The last 19 fields are the genres, a 1 indicates the movie is of that genre, a 0 indicates it is not; movies can be in several genres at once.

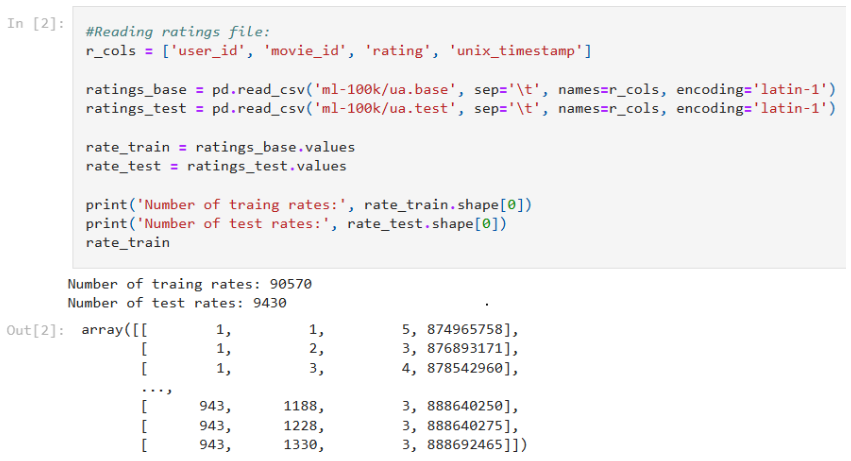
     The movie ids are the ones used in the **`u.data`** file.

- **`u.genre`** – A list of the genres.

2. **File Descriptions**

   - **`u.user`** – Demographic information about the users; this is a tab-separated list of `user id | age | gender | occupation | zip code`. The user IDs are the ones used in the `u.data` data set.
   - **`u.occupation`** – A list of the occupations.
   - **`u1.base`, `u1.test`, ..., `u5.base`, `u5.test`** – The data sets `u1.base` and `u1.test` through `u5.base` and `u5.test` are 80%/20% splits of the `u.data` into training and test data. Each of `u1`, ..., `u5` have disjoint test sets; this is for 5-fold cross-validation (where you repeat your experiment with each training and test set and average the results). These data sets can be generated from `u.data` by `mku.sh`.
   - **`ua.base`, `ua.test`, `ub.base`, `ub.test`**– The data sets `ua.base`, `ua.test`, `ub.base`, and `ub.test` split the `u.data` into a training set and a test set with exactly 10 ratings per user in the test set. The sets `ua.test` and `ub.test` are disjoint. These data sets can be generated from `u.data` by `mku.sh`.
   - **`allbut.pl`** – The script that generates training and test sets where all but `n` of a user's ratings are in the training data.

3. **Splitting Data**



```
In [2]:
#Reading ratings file:
r_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']

ratings_base = pd.read_csv('ml-100k/ua.base', sep='\t', names=r_cols, encoding='latin-1')
ratings_test = pd.read_csv('ml-100k/ua.test', sep='\t', names=r_cols, encoding='latin-1')

rate_train = ratings_base.values
rate_test = ratings_test.values

print('Number of traing rates:', rate_train.shape[0])
print('Number of test rates:', rate_test.shape[0])
rate_train
```

```
Number of traing rates: 90570
Number of test rates: 9430

Out[2]: array([[        1,         1,         5, 874965758],
               [        1,         2,         3, 876893171],
               [        1,         3,         4, 878542960],
               ...,
               [      943,      1188,         3, 888640250],
               [      943,      1228,         3, 888640275],
               [      943,      1330,         3, 888692465]])
```

Figure 4: Split data

The dataset used in this experiment is the MovieLens 100K dataset, which includes user ratings for movies. It was split into two parts: the training data (ua.base) containing 90,570 ratings, and the test data (ua.test) containing

9,430 ratings. The training data is used to build and train the model, while the test data is used to evaluate its performance on unseen ratings. The data was read using Python's pandas library with tab-separated values, and then converted into NumPy arrays for efficient computation. This split ensures that the model is tested for generalization and not just for memorization of the training data, providing a reliable estimate of its performance on new, unseen data.

4.3 Model Training - Algorithms

There are used to 3 Algorithms

1. **Content - Based Recommendations**

Problem: Suppose the number of ***users*** is N, the number of ***items*** is M, and the ***utility matrix*** is represented by matrix $\mathbf{Y}$. The entry in row m, column n of $\mathbf{Y}$ is the level of interest (e.g., rating) of ***user*** n for ***item*** m, which the system has collected. The matrix $\mathbf{Y}$ is sparse and contains many missing entries corresponding to values the system needs to predict.

Additionally, let $\mathbf{R}$ be the *rated or not* matrix, which indicates whether a *user* has rated an *item* or not. Specifically, r_{ij} equals 1 if item i has been rated by user j, and equals 0 otherwise.

Linear Model

Suppose we can find a model for each ***user***, represented by a column vector of coefficients $\mathbf{w}_n$ and a bias term b_n, such that the level of interest of a ***user*** for an ***item*** can be calculated by a linear function:

$$y_{mn} = \mathbf{x}_m \mathbf{w}_n + b_n \tag{4.1}$$

(Note that $\mathbf{x}_m$ is a row vector and $\mathbf{w}_n$ is a column vector.)

For a given *user* n, let the training set be the set of observed entries in $\mathbf{y}_n$. We can construct the loss function similar to *Ridge Regression* as follows:

$$L_n = \frac{1}{2} \sum_{m:r_{mn}=1} (\mathbf{x}_m \mathbf{w}_n + b_n - y_{mn})^2 + \frac{\lambda}{2} \|\mathbf{w}_n\|_2^2 \tag{4.2}$$

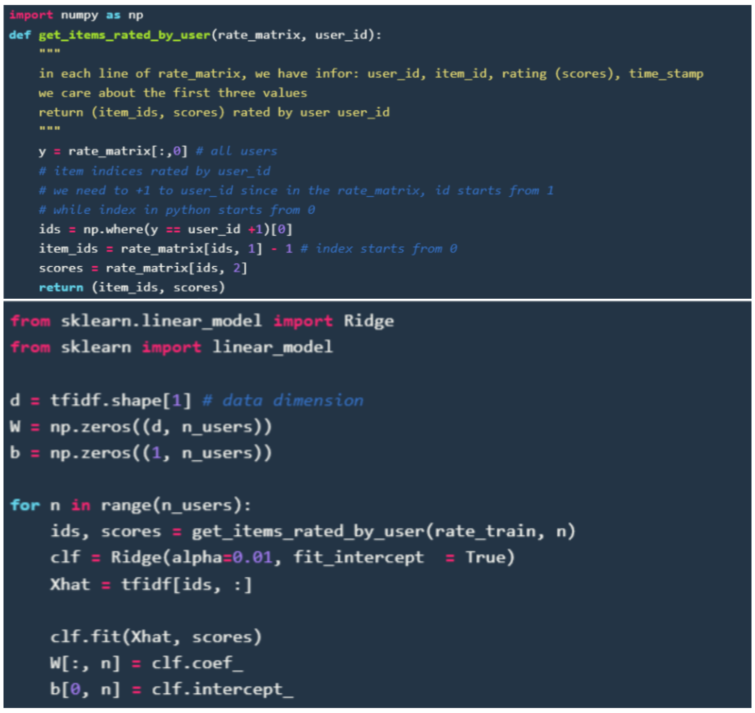
Here, the second term is the *regularization term*, with λ being a positive hyperparameter. Note that regularization is usually not applied to b_n. In practice, the average error is often used, and the loss L_n can be rewritten as:

$$L_n = \frac{1}{2s_n} \sum_{m:r_{mn}=1} (\mathbf{x}_m \mathbf{w}_n + b_n - y_{mn})^2 + \frac{\lambda}{2s_n} \|\mathbf{w}_n\|_2^2 \tag{4.3}$$

Where s_n is the number of *items* that user n has rated. In other words:

$$s_n = \sum_{m=1}^{M} r_{mn}, \tag{4.4}$$

Which is the total of the elements in column n of the *rated or not* matrix $\mathbf{R}$.



```python
import numpy as np
def get_items_rated_by_user(rate_matrix, user_id):
    """
    in each line of rate_matrix, we have infor: user_id, item_id, rating (scores), time_stamp
    we care about the first three values
    return (item_ids, scores) rated by user user_id
    """
    y = rate_matrix[:,0] # all users
    # item indices rated by user_id
    # we need to +1 to user_id since in the rate_matrix, id starts from 1
    # while index in python starts from 0
    ids = np.where(y == user_id +1)[0]
    item_ids = rate_matrix[ids, 1] - 1 # index starts from 0
    scores = rate_matrix[ids, 2]
    return (item_ids, scores)
```

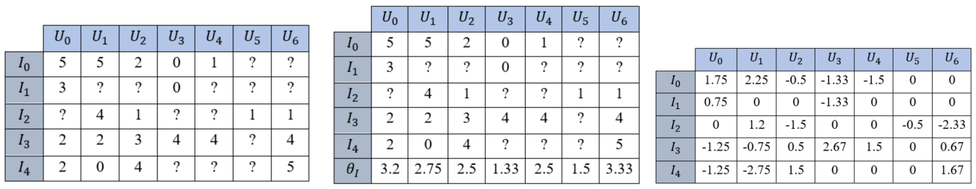
```python
from sklearn.linear_model import Ridge
from sklearn import linear_model

d = tfidf.shape[1] # data dimension
W = np.zeros((d, n_users))
b = np.zeros((1, n_users))

for n in range(n_users):
    ids, scores = get_items_rated_by_user(rate_train, n)
    clf = Ridge(alpha=0.01, fit_intercept  = True)
    Xhat = tfidf[ids, :]

    clf.fit(Xhat, scores)
    W[:, n] = clf.coef_
    b[0, n] = clf.intercept_
```

Figure 5: Model in Python

2. **Neighborhood - Collaborative Filtering**

   Problem: Content-based approaches ignore the preferences and behaviors of other users. This means they miss out on potentially valuable insights that collaborative filtering systems leverage. For instance, a collaborative filtering system might recommend a popular book liked by people with similar preferences, but a content-based system won't. There are 2 branch **User - User collaborative Filtering** and **Item - Item collaborative Filtering**

   User - User collaborative Filtering (UUCF): The input of the problem is the Utility Matrix and the most important job in UUCF must be determined to be the same between two Users and the way to determine similarity is based on the interest level of the items between Users. For example:



	U_0	U_1	U_2	U_3	U_4	U_5	U_6
I_0	5	5	2	0	1	?	?
I_1	3	?	?	0	?	?	?
I_2	?	4	1	?	?	1	1
I_3	2	2	3	4	4	?	4
I_4	2	0	4	?	?	?	5

	U_0	U_1	U_2	U_3	U_4	U_5	U_6
I_0	5	5	2	0	1	?	?
I_1	3	?	?	0	?	?	?
I_2	?	4	1	?	?	1	1
I_3	2	2	3	4	4	?	4
I_4	2	0	4	?	?	?	5
θ_I	3.2	2.75	2.5	1.33	2.5	1.5	3.33

	U_0	U_1	U_2	U_3	U_4	U_5	U_6
I_0	1.75	2.25	-0.5	-1.33	-1.5	0	0
I_1	0.75	0	0	-1.33	0	0	0
I_2	0	1.2	-1.5	0	0	-0.5	-2.33
I_3	-1.25	-0.75	0.5	2.67	1.5	0	0.67
I_4	-1.25	-2.75	1.5	0	0	0	1.67

Figure 6: User similarly matrix

So to conveniently calculate similarity between Users, we have to fill in the values in the ”?” boxes. So what value should each ”?” box be filled in to be appropriate? Many people may think of 0, but I think this will have a big impact on making suggestions because 0 means this is a rather negative level for the item. A value that I find more promising in this case is the average value of the ratings that the User has rated for the remaining items. For example, for the first user, we can calculate this average value to be 3.25, calculated from the average of 5-3-2-2. We will calculate the utility matrix

Item - Item collaborative (IICF): Key characteristics of the user-user CF model:

+ Relies on user similarity, using a ”Similarity matrix” to capture common user preferences. Enables relevant recommendations for less active users.

+ Utility Y method is sparse, requiring limited user rating data to produce relevant recommendations, based on similar users rating items similarly.

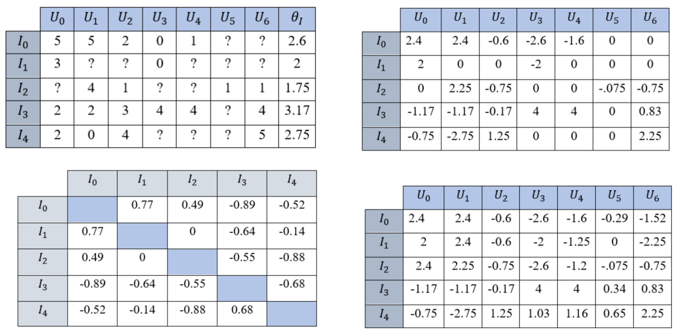
+ Similarity model can complement Utility matrix by recommending items based on proximity to user preferences.

Additional points

+ Similarity matrix captures user similarities, enabling recommendations for less active users.

+ With sparse Utility matrix, Similarity matrix can improve recommendations by leveraging similar user preferences.

Item-item Collaborative Filtering is presented as a complementary approach. For example



	U_0	U_1	U_2	U_3	U_4	U_5	U_6	θ_I
I_0	5	5	2	0	1	?	?	2.6
I_1	3	?	?	0	?	?	?	2
I_2	?	4	1	?	?	1	1	1.75
I_3	2	2	3	4	4	?	4	3.17
I_4	2	0	4	?	?	?	5	2.75

	U_0	U_1	U_2	U_3	U_4	U_5	U_6
I_0	2.4	2.4	-0.6	-2.6	-1.6	0	0
I_1	2	0	0	-2	0	0	0
I_2	0	2.25	-0.75	0	0	-.075	-0.75
I_3	-1.17	-1.17	-0.17	4	4	0	0.83
I_4	-0.75	-2.75	1.25	0	0	0	2.25

	I_0	I_1	I_2	I_3	I_4
I_0		0.77	0.49	-0.89	-0.52
I_1	0.77		0	-0.64	-0.14
I_2	0.49	0		-0.55	-0.88
I_3	-0.89	-0.64	-0.55		-0.68
I_4	-0.52	-0.14	-0.88	0.68	

	U_0	U_1	U_2	U_3	U_4	U_5	U_6
I_0	2.4	2.4	-0.6	-2.6	-1.6	-0.29	-1.52
I_1	2	2.4	-0.6	-2	-1.25	0	-2.25
I_2	2.4	2.25	-0.75	-2.6	-1.2	-.075	-0.75
I_3	-1.17	-1.17	-0.17	4	4	0.34	0.83
I_4	-0.75	-2.75	1.25	1.03	1.16	0.65	2.25

Figure 7: Item similarly matrix

3. **Matrix Factorization**

   Problem: Recall that in **Content-based Recommendation Systems**, each item is described by a vector $\mathbf{x}$, called the *item profile*. In this approach, we aim to find a vector of weights $\mathbf{w}$ corresponding to each user such that the predicted rating y for an item by that user is approximated as:

$$y \approx \mathbf{x}^\top \mathbf{w}$$

   Using this formulation, the **Utility Matrix** $\mathbf{Y}$, assuming all values are filled, is approximated as:

$$\mathbf{Y} \approx \begin{bmatrix} x_1^\top \mathbf{w}_1 & x_1^\top \mathbf{w}_2 & \cdots & x_1^\top \mathbf{w}_N \\ x_2^\top \mathbf{w}_1 & x_2^\top \mathbf{w}_2 & \cdots & x_2^\top \mathbf{w}_N \\ \vdots & \vdots & \ddots & \vdots \\ x_M^\top \mathbf{w}_1 & x_M^\top \mathbf{w}_2 & \cdots & x_M^\top \mathbf{w}_N \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_M^\top \end{bmatrix} \begin{bmatrix} \mathbf{w}_1 & \mathbf{w}_2 & \cdots & \mathbf{w}_N \end{bmatrix} = \mathbf{XW}$$

   where M and N denote the number of *items* and *users*, respectively.

   Suppose we no longer need to predefine the item profiles $\mathbf{x}$, and instead, the feature vectors for each item can be learned simultaneously with the model for each user (here represented by a weight vector). This means that the optimization variables are both $\mathbf{X}$ and $\mathbf{W}$, where:

   + $\mathbf{X}$ is the matrix representing all item profiles, with each row corresponding to one item.

   + $\mathbf{W}$ is the matrix representing all user models, with each column corresponding to one user.

   Using this formulation, we aim to approximate the **Utility Matrix** $\mathbf{Y} \in \mathbb{R}^{M \times N}$

as the product of two matrices:

$$\mathbf{Y} \approx \mathbf{X}\mathbf{W}, \quad \text{with } \mathbf{X} \in \mathbb{R}^{M \times K}, \mathbf{W} \in \mathbb{R}^{K \times N}.$$

Typically, K is chosen to be much smaller than both M and N. In this case, both matrices $\mathbf{X}$ and $\mathbf{W}$ have ranks not exceeding K. Therefore, this method is also referred to as **Low-Rank Matrix Factorization**.

When $\mathbf{X}$ is fixed, optimizing $\mathbf{W}$ becomes the optimization problem in Content-based Recommendation Systems:

$$\mathcal{L}(\mathbf{W}) = \frac{1}{2s} \sum_{n=1}^{N} \sum_{m:r_{mn}=1} \left(y_{mn} - \mathbf{x}_m^\top \mathbf{w}_n\right)^2 + \frac{\lambda}{2} \|\mathbf{W}\|_F^2$$

When $\mathbf{W}$ is fixed, optimizing $\mathbf{X}$ reduces to the optimization problem:

$$\mathcal{L}(\mathbf{X}) = \frac{1}{2s} \sum_{n=1}^{N} \sum_{m:r_{mn}=1} \left(y_{mn} - \mathbf{x}_m^\top \mathbf{w}_n\right)^2 + \frac{\lambda}{2} \|\mathbf{X}\|_F^2$$

CHAPTER 5. NUMERICAL RESULTS

5.1 Evaluation Parameters

RMSE (Root Mean Squared Error):

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

MAE (Mean Absolute Error):

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

TF-IDF

1. (**Term Frequency - Inverse Document Frequency**) measures the importance of a term t in a document d within a corpus. It is calculated as:

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \text{IDF}(t)$$

2. **Term Frequency (TF)**

$$\text{TF}(t,d) = \frac{\text{Frequency of } t \text{ in } d}{\text{Total terms in } d}$$

3. **Inverse Document Frequency (IDF)**

$$\text{IDF}(t) = \log \frac{N+1}{1+\text{DF}(t)} + 1$$

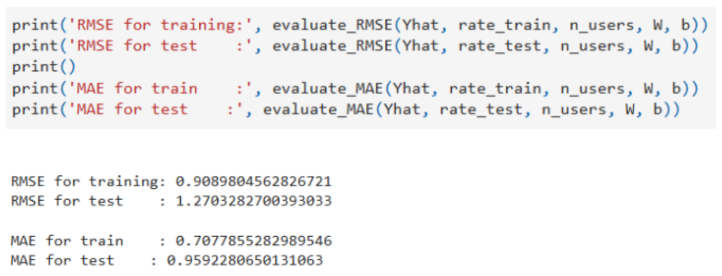
Where:

+ N: Total documents in the corpus.

+ DF(t) : Number of documents containing t

5.2 Result of Evaluating

1. **Content - Based Recommendations:** The Content-Based Recommendation Model shows signs of overfitting, as evidenced by the performance gap between training and test datasets. This suggests the model performs well on familiar data but struggles to generalize to unseen examples. Additionally, the higher RMSE compared to MAE on the test set indicates occasional large prediction errors, reducing its reliability. Overall, while the model may effectively capture individual item-user relationships, its generalization ability and robustness remain limitations, likely requiring enhanced regularization, better feature engineering, or a shift to more versatile recommendation approaches such as hybrid systems.



```
print('RMSE for training:', evaluate_RMSE(Yhat, rate_train, n_users, W, b))
print('RMSE for test    :', evaluate_RMSE(Yhat, rate_test, n_users, W, b))
print()
print('MAE for train    :', evaluate_MAE(Yhat, rate_train, n_users, W, b))
print('MAE for test    :', evaluate_MAE(Yhat, rate_test, n_users, W, b))
```

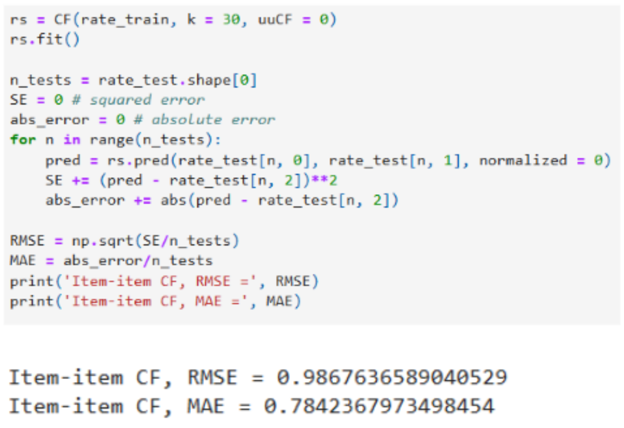
```
RMSE for training: 0.9089804562826721
RMSE for test    : 1.2703282700393033

MAE for train    : 0.7077855282989546
MAE for test    : 0.9592280650131063
```

Figure 8: Evaluation Metrics for Content-Based Recommendation

2. **Neighborhood - Collaborative Filtering**

   User - User collaborative filtering (UUCF) is better prediction results than **content-based filtering** , indicating CF effectively utilizes user relationships to make accurate recommendations. While *content-based* relies solely on product features, **UUCF** has the ability to explore preferences of similar users, creating more novel recommendations. However, **UUCF** faces issues such as the **cold start** problem and computational complexity when the number of users is large.

Item - Item Collaborative Filtering (IICF) The evaluation compares two collaborative filtering approaches: **UUCF** and **IICF**. The results indicate that **IICF** performs slightly better, with lower RMSE and MAE . This suggests that **IICF** is more effective in reducing both large and overall prediction errors in this dataset.



```
rs = CF(rate_train, k = 30, uuCF = 0)
rs.fit()

n_tests = rate_test.shape[0]
SE = 0 # squared error
abs_error = 0 # absolute error
for n in range(n_tests):
    pred = rs.pred(rate_test[n, 0], rate_test[n, 1], normalized = 0)
    SE += (pred - rate_test[n, 2])**2
    abs_error += abs(pred - rate_test[n, 2])

RMSE = np.sqrt(SE/n_tests)
MAE = abs_error/n_tests
print('Item-item CF, RMSE =', RMSE)
print('Item-item CF, MAE =', MAE)
```

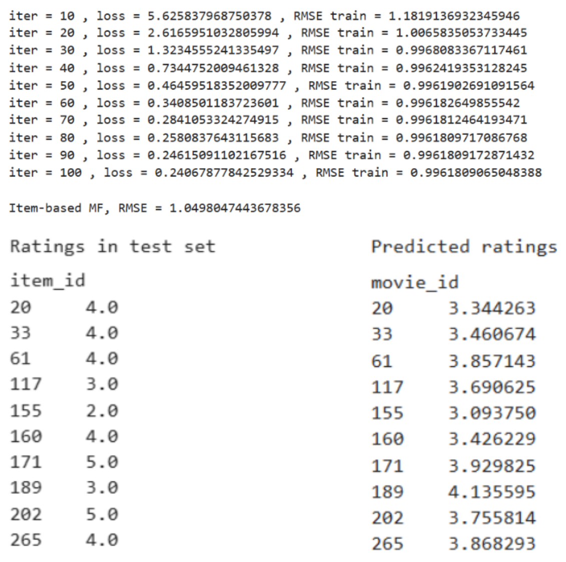
```
Item-item CF, RMSE = 0.9867636589040529
Item-item CF, MAE = 0.7842367973498454
```

Figure 9: Evaluation Metrics for IICF

3. **Matrix Factorization**

GD - Gradient's Descent

The evaluation focuses on the effectiveness of **Matrix Factorization (MF)** using **Gradient Descent** on the MovieLens 100K dataset. Over 100 iterations, the algorithm shows steady convergence, with the loss reducing from 5.63 to 0.24 and the RMSE on the training set decreasing from 1.18 to 0.996. The RMSE on the test set for the **Item-based MF** approach is 1.0498, indicating good predictive performance but slight overfitting. Further optimization with hyperparameters or regularization could improve results. The predicted ratings span a reasonable range, from around 3.09 to 4.14, which aligns well with the actual 5-star rating scale. When comparing the predicted ratings to the ”Ratings in test set”, we can see the model is making reasonably accurate estimates. The Prediction Ratings appear to be providing useful estimates of the true ratings, which is the core purpose of this matrix factorization model

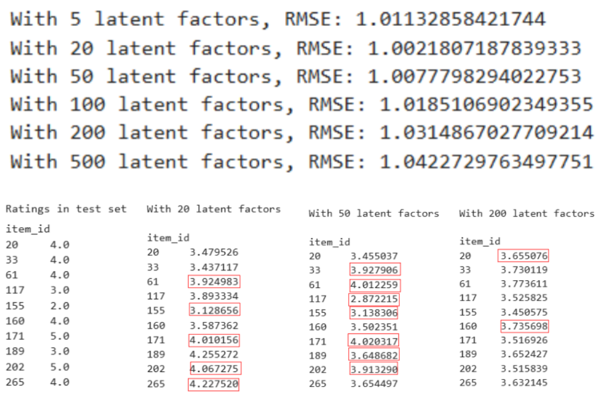


```
iter = 10 , loss = 5.625837968750378 , RMSE train = 1.1819136932345946
iter = 20 , loss = 2.6165951032805994 , RMSE train = 1.0065835053733445
iter = 30 , loss = 1.3234555241335497 , RMSE train = 0.9968083367117461
iter = 40 , loss = 0.7344752009461328 , RMSE train = 0.9962419353128245
iter = 50 , loss = 0.46459518352009777 , RMSE train = 0.9961902691091564
iter = 60 , loss = 0.340850118372360 1 , RMSE train = 0.996182649855542
iter = 70 , loss = 0.2841053324274915 , RMSE train = 0.9961812464193471
iter = 80 , loss = 0.2580837643115683 , RMSE train = 0.9961809717086768
iter = 90 , loss = 0.24615091102167516 , RMSE train = 0.9961809172871432
iter = 100 , loss = 0.24067877842529334 , RMSE train = 0.9961809065048388

Item-based MF, RMSE = 1.0498047443678356

Ratings in test set                     Predicted ratings
item_id                                 movie_id
20      4.0                             20      3.344263
33      4.0                             33      3.460674
61      4.0                             61      3.857143
117     3.0                             117     3.690625
155     2.0                             155     3.093750
160     4.0                             160     3.426229
171     5.0                             171     3.929825
189     3.0                             189     4.135595
202     5.0                             202     3.755814
265     4.0                             265     3.868293
```

Figure 10: Evaluation Metrics for GD

SVD - Singular Value Decomposition

The SVD model shows RMSE values ranging from 1.01 to 1.03 as the number of latent factors is increased from 5 to 500. Comparing this to the Gradient Descent model you previously shared, the RMSE values were generally higher, ranging from around 1.18 to 0.99 as the number of iterations increased. The best RMSE from the Gradient Descent modelat 100 iterations, which is slightly better than the lowest RMSE from the SVD model. So in summary, the SVD model appears to be performing reasonably well, with RMSE values in a similar range to the Gradient Descent approach. The optimal number of latent factors for SVD was found to be 50 for this dataset.



```
With 5 latent factors, RMSE: 1.01132858421744
With 20 latent factors, RMSE: 1.0021807187839333
With 50 latent factors, RMSE: 1.0077798294022753
With 100 latent factors, RMSE: 1.0185106902349355
With 200 latent factors, RMSE: 1.0314867027709214
With 500 latent factors, RMSE: 1.0422729763497751

Ratings in test set   With 20 latent factors   With 50 latent factors   With 200 latent factors
item_id
20      4.0           item_id                  item_id                  item_id
33      4.0           20      3.479526         20      3.455037         20      3.655076
61      4.0           33      3.437117         33      3.927906         33      3.730119
117     3.0           61      3.924983         61      4.012259         61      3.773611
155     2.0           117     3.893334         117     2.872215         117     3.525825
160     4.0           155     3.128656         155     3.138306         155     3.450575
171     5.0           160     3.587362         160     3.502351         160     3.735698
189     3.0           171     4.010156         171     4.020317         171     3.516926
202     5.0           189     4.255272         189     3.648682         189     3.652427
265     4.0           202     4.067275         202     3.913290         202     3.515839
                      265     4.227520         265     3.654497         265     3.632145
```

Figure 11: Evaluation Metrics for SVD

CHAPTER 6. CONCLUSIONS

6.1 Summary

In this paper, a **Movie Recommendation System** is developed using the **MovieLens 100k** dataset, which contains 100,000 movie ratings from 943 users on 1,682 movies. The system implements three key recommendation algorithms:

1. **Content-based Recommendations**: This algorithm recommends movies to users based on the features of the movies and the user's past preferences. Features such as genre, director, and cast are used to calculate similarities between movies.
2. **Neighborhood-based Collaborative Filtering**: This algorithm makes predictions based on the preferences of similar users. It identifies users who have similar movie preferences and recommends movies that those users have rated highly.
3. **Matrix Factorization (e.g., Singular Value Decomposition - SVD)**: Matrix factorization techniques break down the user-item interaction matrix into lower-dimensional matrices representing latent features of users and movies. This method captures hidden patterns in user preferences and movie characteristics that are not immediately apparent from the data.

Dataset and Algorithm Challenges

While the **MovieLens 100k** dataset is a valuable resource, it has some inherent limitations. **Sparsity**: The dataset is sparse, meaning many users have rated only a small fraction of available movies, which makes it difficult to make accurate predictions for unseen items. **Cold Start Problem**: New users or new movies without sufficient ratings make it challenging to generate meaningful recommendations. This issue affects all three algorithms. **Bias in Data**: The dataset contains bias in terms of user activity (some users rate far more movies than others), which can impact the fairness of recommendations.

6.2 Suggestion for Future Works

To enhance the Movie Recommendation System, integrating **Deep Learning** models can help to better capture complex patterns in user preferences and item relationships. Here are several deep learning approaches that can improve the system:

1. **Deep Neural Networks (DNNs)**: DNNs can learn latent representations of users and items by encoding user-item interactions. This can overcome some limitations of traditional methods like matrix factorization, particularly in

capturing non-linear relationships between users and movies.

2. **Autoencoders**: Autoencoders can be used to reconstruct the ratings matrix by learning compressed representations of user-item interactions. This approach helps deal with data sparsity and improves the system's ability to generalize over unseen ratings.
3. **Recurrent Neural Networks (RNNs)**: For sequential recommendations, RNNs, particularly **LSTMs** and **GRUs**, can capture the temporal dynamics of a user's preferences, learning from the order of previously watched movies to predict what the user might want to watch next.
4. **Deep Reinforcement Learning (DRL)**: DRL treats the recommendation task as an optimization problem, where the system learns to make decisions based on user interactions and feedback. This approach is useful for continuously improving recommendations based on real-time user engagement.
5. **Hybrid Models with Neural Collaborative Filtering (NCF)**: By combining traditional collaborative filtering techniques with deep learning models, such as DNNs, hybrid models can capture both linear and non-linear user-item interactions, offering more personalized and effective recommendations.

Benefits of Deep Learning Models

1. **Improved Accuracy**: Deep learning can capture more complex patterns, leading to more precise recommendations.
2. **Better Adaptability**: Models like RNNs and DRL can dynamically adapt to changing user preferences over time.
3. **Scalability**: Deep learning models can scale to large datasets, making them suitable for systems with millions of users and items.

Challenges

1. **Computational Complexity**: Training deep learning models can be resource-intensive, requiring significant computational power.
2. **Data Requirements**: These models need large datasets to train effectively and avoid overfitting.
3. **Interpretability**: Deep learning models, particularly DNNs and GNNs, are often seen as "black boxes" and are harder to interpret compared to traditional recommendation methods.

REFERENCE

Vu Huu Tiep provides the ”Machine Learning Co Ban” website accessed at https://machinelearningcoban.com/.

The GeeksforGeeks article on ”What are Recommender Systems?” accessed at https://www.geeksforgeeks.org/what-are-recommender-systems/.

The Wikipedia article on ”Recommender Systems” at https://en.wikipedia.org/wiki/Recommender_system

Roughgarden, T., & Valiant, G. (2024). CS168: The Modern Algorithmic Toolbox Lecture #9: The Singular Value Decomposition (SVD) and Low-Rank Matrix Approximations., accessed at https://web.stanford.edu/class/cs168/l/l9.pdf/.

Roughgarden, T., & Valiant, G. (2024). CS168: The Modern Algorithmic Toolbox Lecture #9: The Singular Value Decomposition (SVD) and Low-Rank Matrix Approximations., accessed at https://web.stanford.edu/class/cs168/l/l9.pdf.

Rahul Makhijani, Saleh Samaneh, Megh Mehta. CS229 Lecture Discussion: Collaborative Filtering Recommender Systems, accessed at https://cs229.stanford.edu/proj2014/Rahul%20Makhijani,%20Saleh%20Samaneh,%20Megh%20Mehta,%20Collaborative%20Filtering%20Recommender%20Systems.pdf

Zheng Wen. CS229 (2008): Recommendation System Based on Collaborative Filtering, accessed at https://cs229.stanford.edu/proj2008/Wen-RecommendationSystemBasedOnCollaborativeFiltering.pdf