Recommendation system

Fu Yicheng Loke Lam King Ho Jorry Lo Wan Chung

Wong Ching Lok Tong Kin Nam Chu Ki Yu

Group 4

Contents

1 Introduction

The phrase, Personalized recommender systems, has become popular in recent years which aims to provide exact and sufficient recommendation on items to users on the basis of historical information. We built three model of recommendation system indifferent perspectives, which are content-based, item-based collaborative, user-based collaborative filtering. The first model, which is content-based filtering, will treat each users independently and recommend other similar items to each user according their own profile. Unlike Content-based recommendation, item-based collaborative filtering is one of the CF methods to look for similar items based on the items users have already liked or positively interacted with. Lastly, the third model is based on both Graph Neural Network (GNN) and collaborative filtering (CF) as the GNN is proved to be very effective on Recommender System problems by many existing research papers under the rapid development of deep learning.

2 Model 1: Content-Based filtering

2.1 Methodology

The model using content-based filtering is based on linear regression. Before setting up the loss function, we have to define a prediction function, which is parameterized by $w \in \mathbb{R}^{f+1}$, where f the number of genres of the movie in the data set. In the following, we will define all details of our prediction function and also the loss function.

Let the number of users, movies and genres in the data set be u, m and f respectively. Let $R \in \mathbb{R}^{m \times u}$ be the rating matrix, the i^{th} row of R is the rating of the i^{th} movie and the j^{th} column is the rating given by the j^{th} user. Let $\mathbf{R}_i$ be the column vector of R and $\mathbf{R}_i^{(j)} \in \mathbb{R}$ be the j^{th} element of $\mathbf{R}_i$.

Note that for each i^{th} user, $\mathbf{R}_i^{(j)}$ are not well-defined for all $j = 1, 2, 3 \ldots, m$ as the i^{th} users may not give any rating to some movies. Therefore, for each $i = 1, 2, \ldots, u$, we define the set S_i as the following,

$$S_i = \{j \mid \text{if } R_i^{(j)} \text{ is well-defined}\}$$

Let $\mathbf{F}_i \in \mathbb{R}^{f+1}$ be the genre vector of i^{th} movie for $i = 1, 2, \ldots, m$. $\mathbf{F}_i$ indicate the feature in the i^{th} movie, and there are 19 genre in the data set, so $f = 19$ in our project. Also, the first element of $\mathbf{F}_i$ is always be 1, which is a basis term.

After defining all the notations that we need, we can formulate our prediction function and the loss function as follow. Since the model treat each user independently, we can train the parameter of each user individually. For each i^{th} user, we define the prediction function of the rating of the j^{th} movie, which is parameterized by $\mathbf{w}_i \in \mathbb{R}^{f+1}$, $h_{\mathbf{w}_i}(\mathbf{F}_j) : \mathbb{R}^{f+1} \to \mathbb{R}$ as follow,

$$h_{\mathbf{w}_i}(\mathbf{F}_j) = < \mathbf{w}_i, \mathbf{F}_j >$$

, where $< \cdot, \cdot >$ is the dot product of two vector. Although the value of rating is discrete, we will treat the predicted rating as continuous value.

For each i^{th} user, we train the parameter $\mathbf{w}_i$ by minimizing the following loss function using gradient descent,

$$\min_{\mathbf{w}_i} \quad l(h_{\mathbf{w}_i}(\mathbf{F}_j), \mathbf{R}_i) = \min_{\mathbf{w}_i} \quad \frac{1}{2}\sum_{j \in S_i}(h_{\mathbf{w}_i}(\mathbf{F}_j) - \mathbf{R}_i^{(j)})^2$$

By using gradient descent, we can update $\mathbf{w}_i$ at k^{th} iteration as following

$$\mathbf{w}_i^{(k+1)} = \mathbf{w}_i^{(k)} - \alpha \sum_{j \in S_i}(h_{\mathbf{w}_i^{(k)}}(\mathbf{F}_j) - \mathbf{R}_i^{(j)})\mathbf{F}_j$$

,where α is the step size of the gradient descent.

After we found the optimal solution $\mathbf{w}_i^*$ for each $i = 1, 2, 3, \ldots, u$, then we can predict a new movie, which given the genre vector $\mathbf{F}$ of such movie, by using our prediction function $h_{\mathbf{w}_i}(\mathbf{F})$. However, since the minimum value and maximum value of rating is 0.5 and 5 respectively, we need to adjust our prediction p as follow

$$p = \begin{cases} 0.5 & \text{if } h_{\mathbf{w}_i}(\mathbf{F}) \leq 0.5, \\ 5 & \text{if } h_{\mathbf{w}_i}(\mathbf{F}) \geq 5, \\ h_{\mathbf{w}_i}(\mathbf{F}) & \text{else} \end{cases} \tag{1}$$

2.2 Data Processing

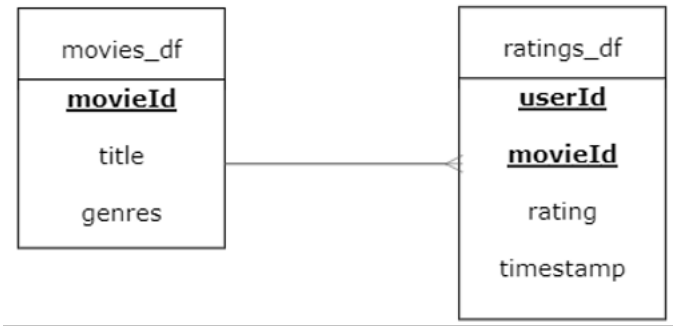


Figure 1: We mainly use the above 2 dataframe in this report

The dataframe "movies_df" stores the genres of each movie, with Column "movieId" as its primary key, while "ratings_df" stores movie ratings from each users, with Column "UserId" and "movieId" as its primary keys.

There are 19 different genres in "movies_df", but we found that there is an irrelevant genre: "(no genres listed)". Here is a figure of "genre_set" , which contains all genres of all movies.

```
{'(no genres listed)',
 'Action',
 'Adventure',
 'Animation',
 'Children',
 'Comedy',
 'Crime',
 'Documentary',
 'Drama',
 'Fantasy',
 'Film-Noir',
 'Horror',
 'IMAX',
 'Musical',
 'Mystery',
 'Romance',
 'Sci-Fi',
 'Thriller',
 'War',
 'Western'}
```

Figure 2: Genres set before data cleaning

Luckily, there are only 34 movies have no genre. Due to the fact that our model cannot work without a proper genre, we decided to remove those 34 movies, with an impact size of 0.35%, which obviously is extremely small. Then the number of rows reduces from 9742 to 9708.

```
print("Before Processing:",movies_df.shape)
movies_df.drop(index=no_genre_index, inplace=True)
print("After Processing:",movies_df.shape)
```

```
Before Processing: (9742, 3)
After Processing: (9708, 3)
```

Figure 3: Shape of movie_df before and after data processing

Then, as movies_df and ratings_df are in one-to-many relationship, we have to do the same to ratings_df by removing the "(no genres listed)" genre category, again, the impact size is very small with only 0.047%.

```
print("Before Processing:",ratings_df.shape)
ratings_df.drop(ratings_df[ratings_df['movieId'].isin(no_genre_movieId)].index, inplace=True)
print("After Processing:",ratings_df.shape)
```

```
Before Processing: (100836, 4)
After Processing: (100789, 4)
```

Figure 4: Shape of rating_df before and after data processing

We need to create a matrix between the userId and movieId with ratings in the next step. However, the number of unique movies in both dataframe does not match, with a difference of 18, as some movies do not have any ratings. We can see this at the following figure.

To facilitate on creating a full matrix and avoid errors, we inserted those movie that do not have any rating back to the rating_df, which have the value of rating as NaN, so that the

```
movies_df['movieId'].nunique()
```

```
9708
```

```
ratings_df['movieId'].nunique()
```

```
9690
```

Figure 5: Number of movies in different Dataframe

number of movies between two dataframe can be match . Afterward, we create a pivot table, which have 9708 movies as row and 610 users as columns, and shown in following figure.

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	NaN	NaN	NaN	4.0	NaN	4.5	NaN	NaN	NaN	...	4.0	NaN	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0
2	NaN	NaN	NaN	NaN	NaN	4.0	NaN	4.0	NaN	NaN	...	NaN	4.0	NaN	5.0	3.5	NaN	NaN	2.0	NaN	NaN
3	4.0	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
193581	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193583	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193585	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193587	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193609	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

9708 rows × 610 columns

Figure 6: Rating matrix

The following step is creating a feature matrix for our model, which is a matrix between genres and movieId. The column of the matrix is movieId and row is the those 19 different genres in our genre set. A movie with corresponding genres will given a '1' value (True), while '0' will given to the false genres of that movieId, which shown in following figure.

	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573	193579	193581	193583	193585	193587	193609
Action	0	0	0	0	0	1	0	0	1	1	...	1	0	0	0	0	1	0	0	1	0
Drama	0	0	0	1	0	0	0	0	0	0	...	0	1	1	0	0	0	0	1	0	0
Mystery	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
Horror	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
Musical	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
IMAX	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
Fantasy	1	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	1	1	0	0	0
Comedy	1	0	1	1	1	0	1	0	0	0	...	1	0	1	0	0	1	1	0	0	1
Children	1	1	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	0
Crime	0	0	0	0	0	1	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
Thriller	0	0	0	0	0	1	0	0	0	1	...	0	0	0	0	0	0	0	0	0	0
Film-Noir	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
Documentary	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1	0	0	0	0	0
Adventure	1	1	0	0	0	0	0	1	0	1	...	0	0	0	0	0	0	0	0	0	0
Animation	1	0	0	0	0	0	0	0	0	0	...	1	1	0	1	0	1	1	0	1	0
Sci-Fi	0	0	0	0	0	0	0	0	0	0	...	1	0	0	0	0	0	0	0	0	0
War	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
Romance	0	0	1	1	0	0	1	0	0	0	...	0	0	0	0	0	0	0	0	0	0
Western	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

Figure 7: Feature matrix

As number of genres are different from each movies, for example, Short Term 12 (2013) only has 1 genre, while Fire with Fire (2012) has 3 genres. Therefore, there is a need to normalize the whole table to ensure the sum of each column equals to 1.

	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573	193579	193581	193583	193585	193587	193609
Thriller	0.0	0.000000	0.0	0.000000	0.0	0.333333	0.0	0.0	0.0	0.333333	...	0.00	0.0	0.0	0.0	0.0	0.00	0.000000	0.0	0.0	0.0
Western	0.0	0.000000	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.000000	...	0.00	0.0	0.0	0.0	0.0	0.00	0.000000	0.0	0.0	0.0
Comedy	0.2	0.000000	0.5	0.333333	1.0	0.000000	0.5	0.0	0.0	0.000000	...	0.25	0.0	0.5	0.0	0.0	0.25	0.333333	0.0	0.0	1.0
Musical	0.0	0.000000	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.000000	...	0.00	0.0	0.0	0.0	0.0	0.00	0.000000	0.0	0.0	0.0
IMAX	0.0	0.000000	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.000000	...	0.00	0.0	0.0	0.0	0.0	0.00	0.000000	0.0	0.0	0.0
War	0.0	0.000000	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.000000	...	0.00	0.0	0.0	0.0	0.0	0.00	0.000000	0.0	0.0	0.0
Animation	0.2	0.000000	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.000000	...	0.25	0.5	0.0	1.0	0.0	0.25	0.333333	0.0	0.5	0.0
Drama	0.0	0.000000	0.0	0.333333	0.0	0.000000	0.0	0.0	0.0	0.000000	...	0.00	0.5	0.5	0.0	0.0	0.00	0.000000	1.0	0.0	0.0
Crime	0.0	0.000000	0.0	0.000000	0.0	0.333333	0.0	0.0	0.0	0.000000	...	0.00	0.0	0.0	0.0	0.0	0.00	0.000000	0.0	0.0	0.0
Action	0.0	0.000000	0.0	0.000000	0.0	0.333333	0.0	0.0	1.0	0.333333	...	0.25	0.0	0.0	0.0	0.0	0.25	0.000000	0.0	0.5	0.0
Film-Noir	0.0	0.000000	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.000000	...	0.00	0.0	0.0	0.0	0.0	0.00	0.000000	0.0	0.0	0.0
Documentary	0.0	0.000000	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.000000	...	0.00	0.0	0.0	0.0	1.0	0.00	0.000000	0.0	0.0	0.0
Romance	0.0	0.000000	0.5	0.333333	0.0	0.000000	0.5	0.0	0.0	0.000000	...	0.00	0.0	0.0	0.0	0.0	0.00	0.000000	0.0	0.0	0.0
Adventure	0.2	0.333333	0.0	0.000000	0.0	0.000000	0.0	0.5	0.0	0.333333	...	0.00	0.0	0.0	0.0	0.0	0.00	0.000000	0.0	0.0	0.0
Children	0.2	0.333333	0.0	0.000000	0.0	0.000000	0.0	0.5	0.0	0.000000	...	0.00	0.0	0.0	0.0	0.0	0.00	0.000000	0.0	0.0	0.0
Mystery	0.0	0.000000	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.000000	...	0.00	0.0	0.0	0.0	0.0	0.00	0.000000	0.0	0.0	0.0
Horror	0.0	0.000000	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.000000	...	0.00	0.0	0.0	0.0	0.0	0.00	0.000000	0.0	0.0	0.0
Sci-Fi	0.0	0.000000	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.000000	...	0.25	0.0	0.0	0.0	0.0	0.00	0.000000	0.0	0.0	0.0
Fantasy	0.2	0.333333	0.0	0.000000	0.0	0.000000	0.0	0.0	0.0	0.000000	...	0.00	0.0	0.0	0.0	0.0	0.25	0.333333	0.0	0.0	0.0

Figure 8: Normalized feature matrix

2.3 Model training

Since we need to train the parameter for each user in order to predict the rating of a movie given by such user, we cannot split the user to two parts . Therefore, We train the model by splitting the movieId and split the data set to the training and testing data into 4:1 ratio. Here is the code of splitting the data set as training set and testing set.

```
# train test split
train_index, test_index = train_test_split(featrue_df.columns,
                                           test_size=.2, train_size=.8,
                                           random_state=101)

# training set
feature_matrix_train = featrue_df[train_index].to_numpy().T
feature_matrix_train = np.hstack((np.ones((len(feature_matrix_train), 1),
                                  dtype=int), feature_matrix_train))
rating_matrix_train = rating_movie_table.loc[train_index].to_numpy()

# testing set
feature_matrix_test = featrue_df[test_index].to_numpy().T
feature_matrix_test  = np.hstack((np.ones((len(feature_matrix_test), 1),
                                  dtype=int), feature_matrix_test))
rating_matrix_test = rating_movie_table.loc[test_index].to_numpy()
```

After that, we train the parameter for users by minimizing the loss function which mentioned in Section 2 using gradient descent with $\alpha = 0.001$ and iterate 500 times. The following code is the code of our model.

```
# Train the weighting vector for user
theta_list = np.array([])
f = feature_matrix_train

for j in range(rating_matrix_train.shape[1]):
    y = rating_matrix_train.T[j].reshape((-1,1))
    theta = np.zeros((f.shape[1],1))
    lamb = 0

    # Gradient descent
    # Step size
    alpha = .001

    for i in range(500):
        theta = theta - alpha*((np.nan_to_num(f@theta-y).T@f).T+
                lamb*theta)

    theta_list = np.append(theta_list, theta)

theta_list = theta_list.reshape(rating_matrix_test.shape[1],-1)
```

2.4 Performance of our model

After we train our model by using the training set, we obtain the following parameter for each user.

```
array([[ 4.07007788, -0.11541042,  0.45833903, ...,  0.05251258,
         0.63571451,  0.        ],
       [ 3.4724376 ,  0.        ,  0.        , ...,  0.57667338,
         0.        , -0.02044903],
       [ 2.38330619,  0.5970223 , -0.16116558, ..., -0.41977739,
        -0.14807892,  0.        ],
       ...,
       [ 3.11019451, -0.19014485, -0.05777242, ..., -0.79952817,
         1.0036285 ,  0.87731913],
       [ 2.94919227,  0.01299759,  0.        , ...,  0.24450514,
         0.01299759, -0.00976934],
       [ 3.64049794, -0.48701151,  0.48990455, ...,  0.06383135,
         0.86406556,  0.00722028]])
```

Figure 9: i^{th} columns is the parameter of i^{th} user

By using the above parameters to predict the testing set with the function (1) mentioned in section 2, we obtain the following result With the range of rating from 0.5 to 5, the result is shown in Table 1.

Metrics	Value
RMSE	1.0741
MSE	1.1536
MAE	0.7935

Table 1: Performance of the first model

3 Model 2: Item-Based Collaborative Filtering (IBCF)

3.1 Methodology

There are two main approaches for item-Based Collaborative Filtering which are Memory-based and Model-based approaches. For the Memory-based approach, it uses user rating historical data to compute the similarity between items and find the most similar to recommend unseen items. For Model-based approach, it uses machine learning algorithms to predict users' rating of unrated items. In this project, KNN algorithm with cross validation and hyperparameter tuning will be used.

Type of Similarity. There are several methods for measuring the similarity between items, which are Cosine, Jaccard, Pearson's correlation, Euclidean distance and Mean Square Distance. The detail of different type of similarity will presented in Table3by using the notation in Table 2.

Notation	Description
U	Sets of user
$r_{u,i}$	User u rating value for item i
$r_{u,j}$	User u rating value for item j
U_i	A set of user who have rated for item i
U_j	A set of user who have rated for item j
u_i	Average rating value from user i
u_j	Average rating value from user j

Table 2: Description of notations.

Name	Formula
Cosine Similarity	$cos_{ij} = \frac{\sum_{u \in U} r_{u,i} r_{u,j}}{\sqrt{\sum_{u \in U} r_{u,i}^2} \sqrt{\sum_{u \in U} r_{u,j}^2}}$
Jaccard Similarity	$J(i,j) = \frac{\lvert U_i \cap U_j \rvert}{\lvert U_i \cup U_j \rvert}$
Pearson Correlation similarity	$pearson_{ij} = \frac{\sum_{u \in U} (r_{u,i} - \mu_i)(r_{u,j} - \mu_j)}{\sqrt{\sum_{u \in U} (r_{u,i} - \mu_i)^2} \sqrt{\sum_{u \in U} (r_{u,j} - \mu_j)^2}}$
Euclidean similarity	$\frac{1}{1 + \sqrt{\sum_{u \in U} (r_{u,i} - r_{u,j})^2}}$
Mean Square Difference (MSD) similarity	$msd\_sim(i,j) = \frac{1}{1 + msd(i,j)}$ $msd(i,j) = \frac{1}{\lvert U_{ij} \rvert} \sum_{u \in U_{ij}} (r_{ui} - r_{uj})^2$

Table 3: Formula of different type of similarity.

3.2 GridSearchCV (Grid Search + Cross Validation)

In reality, models often have many parameters to tune for finding the best fitting one for the dataset and use case. GridSearchCV module can be used to find the optimal setting of the parameter of the model, by training the model with parameters over all possible combinations of the parameters, and then evaluate with test data by cross validation technique. This is called N-fold cross validation.

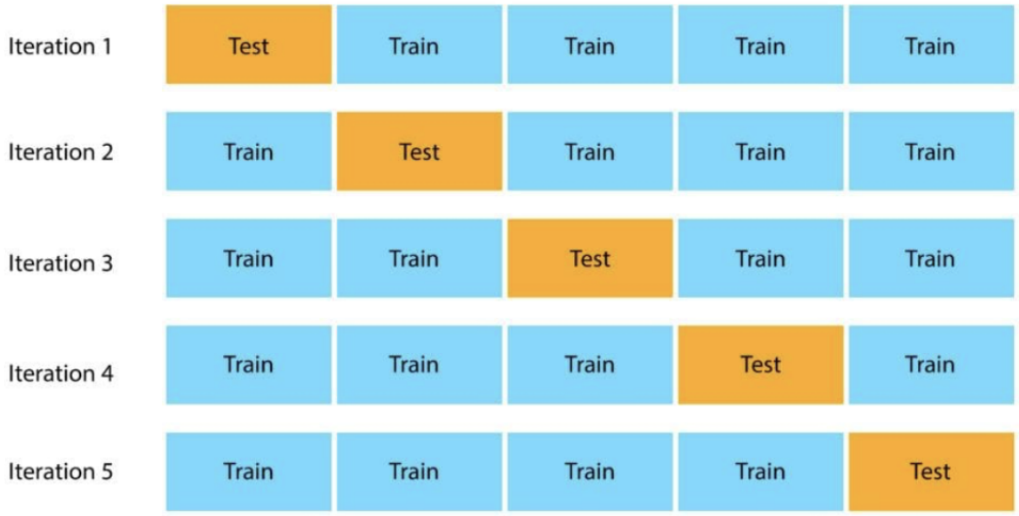


Figure 10: Cross Validation

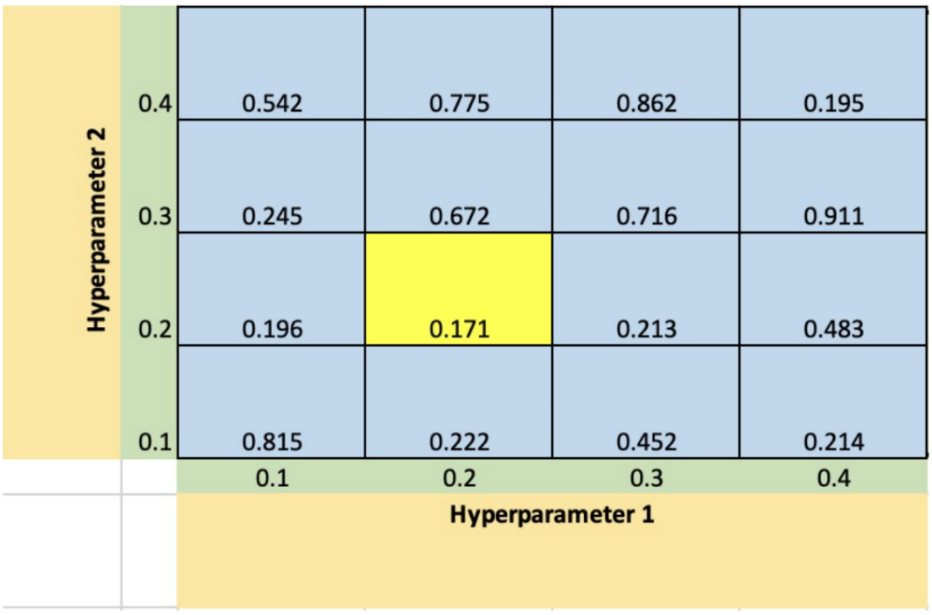


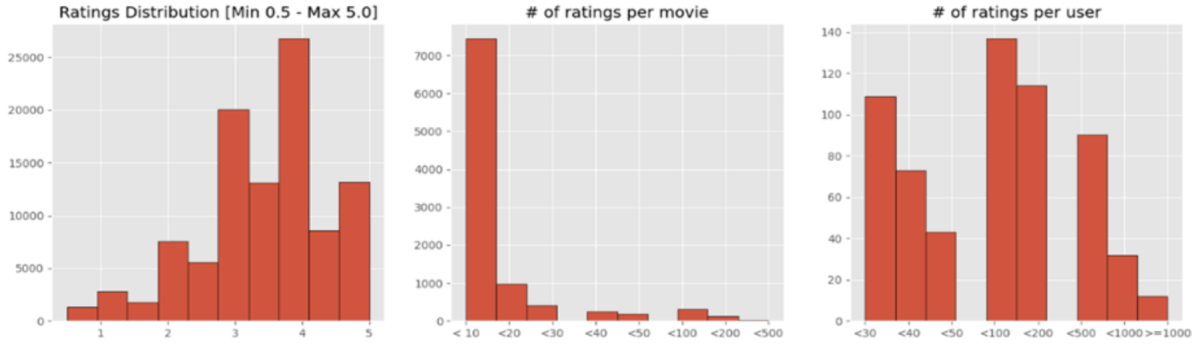
Figure 11: GridSearchCV

3.3 Exploratory analysis

Ratings.csv data is used in both memory-based and model-based approaches.The data consists of 610 users from 9724 movies with 100,836 ratings. For the rating, it ranges from 0.5 to 5. And the sparsity level of this dataset is 98.3%. (only 1.7%user-item known ratings).

```
There are # 100836 rating in ratings.csv
The value in rating is: [0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]
There are # 9724 different value (movieId) in ratings.csv
There are # 610 different value (userId) in ratings.csv
The sparsity level of this dataset is 98.3%
```

From the below graph, there are $\sim$ 7400 movies with below #10 ratings and $\sim$ 180 users who have rated less than 40 movies.



3.4 Data Pre-processing and Transformation

Due to the large sparsity level in data, we have defined valid movie and user based on below criteria to avoid bias due to inactive user and unpopular movies.

- Users who have rated at least 40 movies.
- A movie that has been watched at least 10 times.

After excluding the invalid movie and users, the new data consists of 428 users and 2,269 movies with 76,395 ratings. And the sparsity level of this dataset reduces to 92.1% (7.8% user-item known ratings).

3.5 Memory-based Approach

For Memory-based approach, transformation on variable (rating, movie id and user id) is required for modelling development.

1. **Rating Variable.** For the rating variable, as different users can rate the same item differently depending upon how optimistic they are. The ratings will be normalised by

subtracting the user's average rating (rating_mean) and in order to overcome optimism issues. A new variable "rating_norm" has been created. Apart from that, to calculate the Jaccard similarity, a binary rating variable "rating_binary" will be created based on following criteria:

- Like: 1, if Rating ≥ 3.5
- Dislike: 0, if Rating < 3.5

	userId	movieId	rating	rating_mean	rating_norm	rating_binary
0	1	1	4.0	4.383886	-0.383886	1
1	1	3	4.0	4.383886	-0.383886	1
2	1	6	4.0	4.383886	-0.383886	1
3	1	47	5.0	4.383886	0.616114	1
4	1	50	5.0	4.383886	0.616114	1
5	1	70	3.0	4.383886	-1.383886	0

Figure 12: Example of rating_norm and rating_binary

2. **User Id and Movie Id.** As the userId and MovieId is not in arithmetic sequence after excluded invalid movie and users, new user id and movie will be assigned in below:

   - User Id : from 0 to 427
   - Movie Id: from 0 to 2268

	userId	movieId	rating	rating_mean	rating_norm	rating_binary
0	0	0	4.0	4.383886	-0.383886	1
1	0	2	4.0	4.383886	-0.383886	1
2	0	4	4.0	4.383886	-0.383886	1
3	0	34	5.0	4.383886	0.616114	1
4	0	36	5.0	4.383886	0.616114	1

Figure 13: Assign new user id and movie id in arithmetic sequence

3.5.1 Model Development (Memory-based)

Python Package "sklearn" will be used for train-test split, similarity calculation and accuracy.

1. **Defining Training and Testing set data.** By using train_test_split in scikit-learn, 80% of the data will be used for model development (training data) and 20% of the data will be used for test data. Due to the high sparsity level of data, consistency check is required to ensure the training set and testing set is having the same dimension.

```
#Assign X as the ratings_base dataframe and Y as the movieId column of ratings.
X = ratings_base.copy()
Y = ratings_base['movieId']
#Split into training and test datasets, stratified along movieId
np.random.seed(24)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, stratify = Y)
```

```
def check_consistency(df, name = "Name"):
    # Count the nr. of unique movie and users
    n_users = df['userId'].unique().shape[0]
    n_items = df['movieId'].unique().shape[0]

    print('There are {} unique users and {} unique items in {} dataframe.'.format(n_users, n_items, name))
    print('Min userId = {}, max userId = {} in {} dataframe.'.format(df['userId'].min(), df['userId'].max(), name))
    print('Min movieId = {}, max movieId = {} in {} dataframe.'.format(df['movieId'].min(), df['movieId'].max(), name))
```

```
There are 428 unique users and 2269 unique items in X_Train dataframe.
Min userId = 0, max userId = 427 in X_Train dataframe.
Min movieId = 0, max movieId = 2268 in X_Train dataframe.
----------------------------------------------------------------
There are 428 unique users and 2269 unique items in X_Test dataframe.
Min userId = 0, max userId = 427 in X_Test dataframe.
Min movieId = 0, max movieId = 2268 in X_Test dataframe.
```

2. **Converting data into utility matrix (user-movie matrix).** To calculate the similarity between items, both train and test data will convert into array or csr format.

```
def transform_to_csr_array(df, rating='rating'):
    df_ratings_pivot = df.pivot(index='userId', columns='movieId', values=rating)
    df_ratings_pivot_dummy = df_ratings_pivot.copy().fillna(0)
    rating_csr = csr_matrix(df_ratings_pivot_dummy)
    rating_array = rating_csr.toarray()
    return df_ratings_pivot, df_ratings_pivot_dummy, rating_csr, rating_array
```

3. **Computing the similarity matrix and rating prediction.** Sklearn provides 4 methods for calculating similarities, namely Cosine, Jaccard, correlation (Pearson) and Euclidean similarity. After constructing the similarity matrix, we will perform rating prediction to testing data by weighted average approach.

```
def predict(ratings,similarity):
    pred = ratings.dot(similarity)/np.array([np.abs(similarity).sum(axis=1)])
    pred[np.isnan(pred)] = 0
    return pred
```

The Rating Prediction Formula is defined as follow,

$$\hat{r}_{ui} = \frac{\sum_{j \in N_u^k(i)} sim(i,j) r_{uj}}{\sum_{j \in N_u^k(i)} |sim(i,j)|}$$

4. **Model Performance and recommendation.** RMSE will be used for the accuracy measurement. To calculate the RMSE, we compare the prediction value to non-zero value in the test dataset.

```python
from sklearn.metrics import mean_squared_error
from math import sqrt
def rmse(prediction,ground_truth):
    prediction = prediction[ground_truth.nonzero()].flatten()
    ground_truth = ground_truth[ground_truth.nonzero()].flatten()
    return sqrt(mean_squared_error(prediction,ground_truth))
```

By comparing the RMSE in the testing set, the best model is cosine similarity with normalised rating, which shown in Table 4

Type of Similarities	**Train RMSE**	**Test RMSE**
Cosine (Original)	3.06897	3.15300
Cosine(Normalised)	0.82231	0.89639 (Best Model)
Pearson (Original)	3.39722	3.51834
Pearson (Normalised)	0.90371	0.91279
Euclidean (Original)	3.26431	3.31320
Euclidean (Normalised)	0.91255	0.91582
Jaccard	0.82915	0.90611

Table 4: RMSE with different type of similarities

Below are the top 10 recommendations based on the best model, it shows that the recommendation is also reasonable and expected.

```python
def get_top_10_similar_movies(movie_id,similarities ):
    top_10_similar_movies = list(np.argsort(similarities[movie_id, :])[:-12:-1])
    top_10_similar_movies.pop(0)
    return top_10_similar_movies
```

```python
def print_10_recommendation(watched_id,similarities):
    cosine_10_movie_id = get_top_10_similar_movies(watched_id,similarities)
    print("Because you watch:",id_to_title_dict[movie_new_to_old_dict[watched_id]])
    print("-------------------------------------")
    print("Below movies you might like:")
    print("                                    ")
    for i in cosine_10_movie_id:
        print(id_to_title_dict[movie_new_to_old_dict[i]])
print_10_recommendation(0,cos_norm_similarities)
```

```
Because you watch: Toy Story (1995)
-----------------------------------
Below movies you might like:

Toy Story 2 (1999)
Toy Story 3 (2010)
Babe (1995)
E.T. the Extra-Terrestrial (1982)
Star Wars: Episode IV - A New Hope (1977)
Hugo (2011)
Blazing Saddles (1974)
Aladdin (1992)
Airplane! (1980)
Willy Wonka & the Chocolate Factory (1971)
```

3.6 Model based Approach - KNN

For the KNN-based Rating Prediction, a package Surprise will be used to model the rating. For detail, please visit: `https://surpriselib.com/`.

1. Load the filtered data in pandas dataframe to surprise.dataset

```
def create_dataset(df):
    reader = Reader(rating_scale=(df.rating.min(), df.rating.max()))
    data = Dataset.load_from_df(df[["userId", "movieId", "rating"]], reader)

    return data
✓ 0.2s

data = create_dataset(data_filtered)
✓ 0.5s
```

2. First GridSearch (Roughly searching).
   NOTE: for details of parameter and the equation for calculation, please see appendix.

   (a) Define the parameter grid. In the first search, a less granular grid will be used.

```
param_grid = {
    'k': [i for i in range(10, 100, 10)],
    'sim_options': {
        'name': ['msd', 'cosine', 'pearson'],
        'min_support': [1, 5, 10],
        'user_based': [False],
    },
}
```

(b) Run the gridsearch of parameter grid on 3 different KNN-based model.

```
gs_knn_basic = GridSearchCV(
    KNNBasic, param_grid, measures=["rmse"],
    cv=5, n_jobs=-1, return_train_measures=True
    )
gs_knn_basic.fit(data)
```

```
gs_knn_mean = GridSearchCV(
    KNNWithMeans, param_grid, measures=["rmse"],
    cv=5, n_jobs=-1, return_train_measures=True
    )
gs_knn_mean.fit(data)
```

```
gs_knn_zscore = GridSearchCV(
    KNNWithZScore, param_grid, measures=["rmse"],
    cv=5, n_jobs=-1, return_train_measures=True
    )
gs_knn_zscore.fit(data)
```

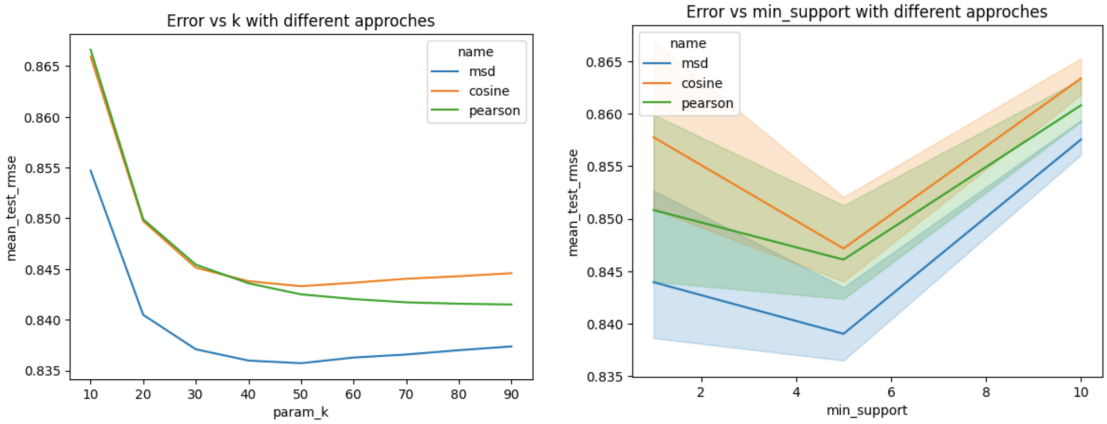
3. Analysis the result from the first GridSearch

```
KNN_Basic
best RMSE score: 0.8649708122685162
best parameters{'k': 20, 'sim_options': {'name': 'msd', 'min_support': 5, 'user_based': False}}

KNN_Mean
best RMSE score: 0.8357177885102682
best parameters{'k': 50, 'sim_options': {'name': 'msd', 'min_support': 5, 'user_based': False}}

KNN_ZScore
best RMSE score: 0.8405009124349482
best parameters{'k': 50, 'sim_options': {'name': 'msd', 'min_support': 5, 'user_based': False}}
```

Since KNN_Mean obtains the lowest RMSE, KNN_Mean is selected to continue the rest of fine tuning.



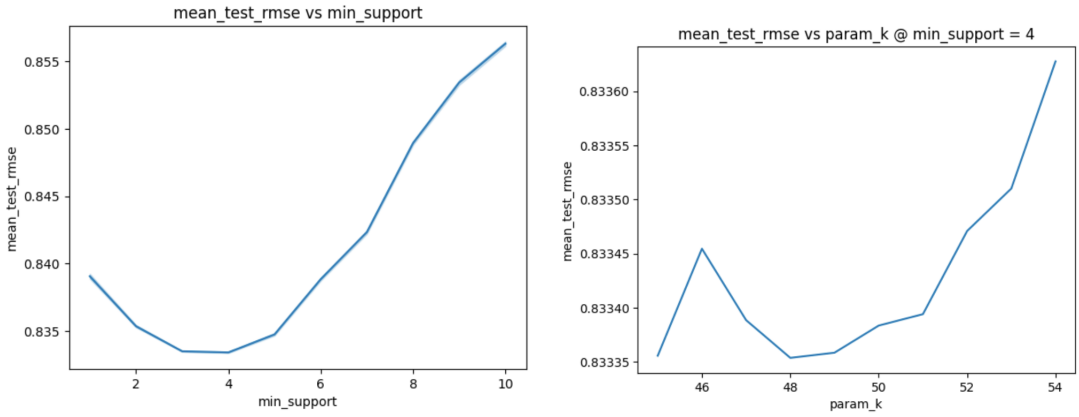
As it is obvious that the msd similarity outperform another two similarity, in the next step of fine tuning, only msd will be used.

4. Further fine-tune the parameter to achieve lowest RMSE.

```
param_grid = {
    'k': [i for i in range(45, 55, 1)],
    'sim_options': {
        'name': ['msd'],
        'min_support': [i for i in range(1, 11)],
        'user_based': [False],
    },
}
```

Result: After the running the gridsearch on above parameter grid, this optimal parameter is obtained:

```
{'rmse': {'k': 48,
  'sim_options': {'name': 'msd', 'min_support': 4, 'user_based': False}}}
```



From the mean_test_rmse vs param_k @ min_support = 4, it is observed that the model has the lowest RMSE at the $K = 48$.

Note: Although the graph is not very smooth (the RMSE suddenly rises at 46), this can be explained due to the scale of the y-axis being very small, some fluctuation of RMSE may be experienced. To further improve this, increasing the number of fold of cross validation will normally smooth this abnormal peak.

Therefore, the best performing model for this set of data will be the KNN_Mean with the following parameters: $K = 48$, Similarity_function = MSD, Min_support = 4.

4 Model 3: Graph Neural Network-Based Collaborative Filtering

A brand-new Model 3 would be introduced in this section which is the Graph Neural Network (GNN)-based collaborative filtering (CF) method on Recommender System (RS). The prominent CF paradigms have changed from matrix factorization (MF) methods in the early time such as conventional matrix factorization methods [3] to neural network-based methods in recent years. Therefore, we apply the GNN-CF in our project beyond the previous two models.

4.1 Graph Neural Network

With the achievement of GNN on graph representations learning, some CF-based recommendation models based on GNN utilize the fact that most of the data in recommendation could form a graph structure. In detail, the key objects in recommendation including users, items, attributes and so on which connect with or influence each other (e.g., user-item interactions) could be well-presented by a graph structure [4]. For example, diverse types of graphs come from the information in recommender systems (knowledge graph and hypergraph) which significantly benefit the effectiveness of the recommender systems [6]. Therefore, GNN are used to capture the crucial CF signals of user-item interactions to enhance the representation quality with the propagation process [11] [8]. Many existing state-of-the-art RS models already prove the significance of GNN. For example, PinSage [9], GC-MC [1] and NGCF [7] all apply the graph convolutional network (GCN) on the user-item interaction graph. LightGCN [2] further adopts non-linear transformation for propagation and applies the summation-related pooling method into neighboring aggregation.

4.2 Data Preprocess

In terms of data processing, no data will be removed to improve the sparsity issue as it is the common difficulty existing in most of the datasets. Instead, this issue can be well-handled by the model. 80% of data will be utilized for training and the model will be tested on the remaining set. The random splitting is conducted on each single user.

4.3 Inductive Graph-based Matrix Completion (IGMC)

Background. IGMC [10] was developed in 2020 which is an inductive matrix completion model without using any side information which is always not available in the raw datasets. It will train a graph neural network merely based on 1-hop subgraphs around each user-movie pair which could be obtained from the rating matrix and assign the subgraphs to the corresponding ratings. Most importantly, it is inductive as the prediction for the new/unseen user-movie pair is available given that their interactions with the other users/movies exist. In IGMC, the local graphs around the target user-movie pair are effective predictors of the corresponding rating.

Hop. 1-hop subgraph refers to a particular user-movie pair and their direct neighbors, e.g., the movies which previously received the ratings from the target user or the users who gave ratings to the target movie before.

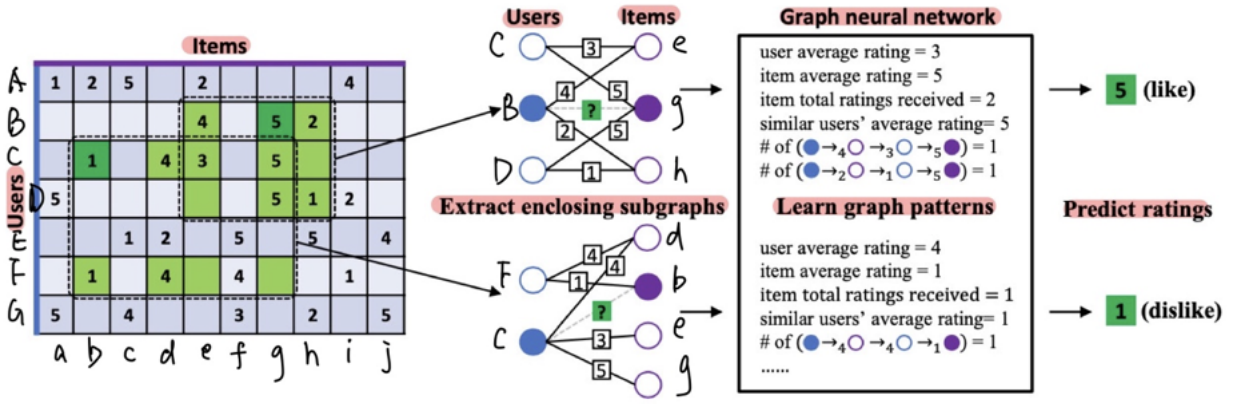


Figure 14: The Flow of IGMC. (1-hop neighbors applied.)

Framework. Figure 14 indicates the workflow of IGMC. A rating matrix is given where 2 dark-green blocks are our target user-movie pairs (User B $\rightarrow$ movie g and User C $\rightarrow$ movie b), and two enclosing subgraphs will be generated for them respectively. Horizontally, the movies to be involved in the 1-hop subgraphs can be identified in the rating matrix (in the row of User B, movies e and h should be included). Similarly, the users to be involved in the 1-hop subgraphs of the target pair can be found vertically in the matrix. Once the subgraphs are completed, they will be fed into a Graph Neural Network (GNN) which learns the possible patterns inside the graph, e.g., the average rating and the number of ratings the target movie received from the similar users, the mean ratings the users gave to the similar movies and so on. Finally, the GNN maps each subgraph to the rating that its center user gives to its center item.

GNN model. With the enclosing subgraph for each pair of (u, v), we train a GNN which maps each subgraph to the rating in two parts: 1) message passing layers which learn a feature vector for each node from the subgraph, and 2) a pooling layer which generates a subgraph representation based on node features. We use the relational graph convolutional operator (R-GCN) [5] as the message passing layers in our GNN model as follows:

$$x_i^{l+1} = W_0^l x_i^l + \sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_r(i)} \frac{1}{|\mathcal{N}_r(i)|} W_r^l x_j^l,$$

where x_i^l represents the feature vector of node i at layer l and W_0^l, W_r^l represent the learnable parameter matrices respectively. From different W_r^l, various graph patterns could be learned, such as similar users' average rating, item average rating and so on. Then, with stacking L message passing layers using tanh activation functions for any two layers, the final representation h_i for feature vectors from different layers of node i is designed under concatenation as:

$$h_i = concat(x_i^1, x_i^2, \ldots, x_i^L)$$

A simple but well-performed pool layer, which outputs the graph representation by concatenating both target user's and item's final representations, is defined with the following formula:

$$g = concat(h_u, h_v),$$

where g refers to the graph representation and h_u, h_v refer to target user's and item's final representations respectively. This simple pooling choice is made due to the extreme significance of these two target nodes compared with others. Finally, a MLP is adopted to generate the predicted rating as follows:

$$\hat{r} = w^T \sigma(Wg),$$

where w, W denote the parameter of MLP outputting a scalar rating $\hat{r}$ from the graph representation g with σ as the ReLU activation function.

4.4 Model Training and Performance Evaluation

We train the model with minimizing the mean square error (MSE) between the ground truth and the predictions for ratings in train set. As the IGMC already provides the fine-tuned hyperparameters from cross validation for ML-100K (which is the origin of our project dataset), we adopt most fine-tuned values directly in our architecture of GNN without using validation set along with some self-tuned values, shown in Table 5.

Name	Description (e.g., number, range, etc.)
R-GCN	Layers: 4 Hidden dimensions: {32,32,32,32} Decomposition bases: 4
MLP	Hidden units: 128 Dropout rate: 0.5
Hop	1-hop (computational time)
Enclosing subgraph	Dropout rate: 0.2
Adam optimizer	Batch size: {20} Initial learning rate: $1e^{-3}$ Decay ratio: 0.1 Epoch: {40}

Table 5: The fine-tuned hyperparameters in IGMC.

Then, we test our model performance by two traditional evaluation methods MSE and Root MSE (RMSE) and one widely-used protocol *recall@K* (i.e., the proportion of positive items included in top-K recommendations, with default $K = 10$), as shown in Table 6. It is remarkable that the RMSE values of our model and IGMC are almost the same, which

further confirms our model's advantages and the *recall*@10 value is also significantly larger than other existing papers' value on MovieLens 100K Dataset.

Model	MSE	RMSE	*recall*@K
IGMC (Baseline)	—	0.905	—
Model 3 (Ours)	0.821	0.906	0.333

Table 6: Test results from our models, previous two models and baseline IGMC. (Remark. — : not applicable; *recall*@K is calculated with ignoring the users which have less than K true given ratings; Final *recall*@K value is computed by averaging the recall values across all users.)

5 Conclusion

In conclusion, there are different kind of recommendation system. Each of them have their own advantage and disadvantage and the method of preprocessing the data is also different. The main disadvantage content-based filtering is that the model cannot operate if we do not know the feature of a movie and we cannot recommend any item to a new user as we do not have the parameter of this new user. However, the model prevent the problem of long-tail of recommendation system, which is the system may not recommend those unpopular item to user, but this model can also predict those unpopular item as long as we have the feature. Without any side information, the third model has shown good performance for the recommendation system tasks compared with the traditional methods (content-based, user-based and item-based) by merely utilizing the local graph structure and the deep learning method. Most importantly, it is inductive which means it generalizes to new data points that are unseen in the training dataset. This advantage makes it a more practical model as the number of new movies is becoming larger and larger in today's life. Last but not least, it makes full use of user-item interactions for prediction which indicates the growing importance of network analysis and deep learning technics on the field of recommendation system.

6 Presentation

Here is the link of our presentation `https://youtu.be/sUi74zTZT9Y`.

References

[1] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[2] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 639–648, 2020.

[3] Chao Huang. Recent advances in heterogeneous relation learning for recommendation. *arXiv preprint arXiv:2110.03455*, 2021.

[4] Chao Huang, Jiahui Chen, Lianghao Xia, Yong Xu, Peng Dai, Yanqing Chen, Liefeng Bo, Jiashu Zhao, and Jimmy Xiangji Huang. Graph-enhanced multi-task learning of multi-level transition dynamics for session-based recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4123–4130, 2021.

[5] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pages 593–607. Springer, 2018.

[6] Shoujin Wang, Liang Hu, Yan Wang, Xiangnan He, Quan Z Sheng, Mehmet A Orgun, Longbing Cao, Francesco Ricci, and Philip S Yu. Graph learning based recommender systems: A review. *arXiv preprint arXiv:2105.06339*, 2021.

[7] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174, 2019.

[8] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: a survey. *ACM Computing Surveys (CSUR)*, 2020.

[9] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983, 2018.

[10] Muhan Zhang and Yixin Chen. Inductive matrix completion based on graph neural networks. *arXiv preprint arXiv:1904.12058*, 2019.

[11] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

A KNN Model Parameters

For all KNN-based models above, they shared a same parameter space, which include:

Parameter Name	Explanations
k	The maximum number of neighbours to consider, the true k will be less than or equal to this number. This is due to the sparsity of the data, which there may not be enough neighbours for the prediction.
similarity	In KNN approach, 3 methods will be considered and tested to find the best fitting similarity method. Which include cosine, pearson, and mean square error, the equation of these similarity can be found in "similarity" session.
min_support	The min_support means the minimum common users for the similarity, if number of common users is below min_support, the similarity will be 0.

B KNN-based Rating Prediction Methodology

There are few different approaches in predicting the ratings,

1. **Basic KNN.** The basic KNN method predicts the rating by using the similarity weighted average rating of K-neighbours.

$$\hat{r} = \frac{\sum_{j \in N_u^k(i)} sim(i,j) r_{uj}}{\sum_{j \in N_u^k(i)} sim(i,j)}$$

2. **KNN with mean** The KNN will take the mean rating of the movie into account, compared to the basic KNN method.

$$\hat{r} = \mu_i + \frac{\sum_{j \in N_u^k(i)} sim(i,j)(r_{uj} - \mu_j)}{\sum_{j \in N_u^k(i)} sim(i,j)}$$

3. **KNN with z score** Just like the KNN with mean, but this time the standard deviation of the movie rating is also considered when predicting.

$$\hat{r} = \mu_i + \sigma_i \frac{\sum_{j \in N_u^k(i)} sim(i,j) \frac{r_{uj} - \mu_j}{\sigma_j}}{\sum_{j \in N_u^k(i)} sim(i,j)}$$

C Contribution Table

Content	Name
Introduction	Equal Contribution for all group members
Model 1	Equal Contribution for Lo Wan Chung and Wong Ching Lok
Model 2	Equal Contribution for Tong Kin Nam and Chu Ki Yu
Model 3	Equal Contribution for Fu Yicheng and Lam King Ho
Conclusion	Equal Contribution for all group members

Table 7: Contribution Table for Report and Presentation

Content	Name
Model 1 (Content-based.ipynb)	Equal Contribution for Lo Wan Chung and Wong Ching Lok
Model 2	Equal Contribution for Tong Kin Nam and Chu Ki Yu
Model 3	Equal Contribution for Fu Yicheng and Lam King Ho

Table 8: Contribution Table for coding