

Project - Recommendation Systems: Movie Recommendation System

Marks: 40

Context

Online streaming platforms like **Netflix** have plenty of movies in their repository and if we can build a **Recommendation System** to recommend **relevant movies** to users, based on their **historical interactions**, this would **improve customer satisfaction** and hence, it will also improve the revenue of the platform. The techniques that we will learn here will not only be limited to movies, it can be any item for which you want to build a recommendation system.

Objective

In this project we will be building various recommendation systems:

- Knowledge/Rank based recommendation system
- Similarity-Based Collaborative filtering
- Matrix Factorization Based Collaborative Filtering

we are going to use the **ratings** dataset.

Dataset

The **ratings** dataset contains the following attributes:

- userId
- movieId
- rating
- timestamp

Sometimes, the installation of the surprise library, which is used to build recommendation systems, faces issues in Jupyter. To avoid any issues, it is advised to use **Google Colab** for this case study.

Let's start by mounting the Google drive on Colab.

```
In [1]: # uncomment if you are using google colab

#from google.colab import drive
#drive.mount('/content/drive')
```

Installing surprise library

```
In [2]: # Installing surprise library, only do it for first time
!pip install surprise
```

Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/public/simple/>
Collecting surprise
 Downloading surprise-0.1-py2.py3-none-any.whl (1.8 kB)
Collecting scikit-surprise
 Downloading scikit-surprise-1.1.3.tar.gz (771 kB)
 772.0/772.0 kB 9.9 MB/s eta 0:00:00
 Preparing metadata (setup.py) ... done
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise->surprise) (1.2.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise->surprise) (1.22.4)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-surprise->surprise) (1.10.1)
Building wheels for collected packages: scikit-surprise
 Building wheel for scikit-surprise (setup.py) ... done
 Created wheel for scikit-surprise: filename=scikit_surprise-1.1.3-cp310-cp310-linux_x86_64.whl size=3095454 sha256=7c43d71895bebfdel743acf9eb7464a2521681cc3c486f65adae8c93f4a72dc0
 Stored in directory: /root/.cache/pip/wheels/a5/ca/a8/4e28def53797fdc4363ca4af740db15a9c2f1595ebc51fb445
Successfully built scikit-surprise
Installing collected packages: scikit-surprise, surprise
Successfully installed scikit-surprise-1.1.3 surprise-0.1

Importing the necessary libraries and overview of the dataset

```
In [3]: # Used to ignore the warning given as output of the code
import warnings
warnings.filterwarnings('ignore')

# Basic libraries of python for numeric and dataframe computations
import numpy as np
import pandas as pd

# Basic library for data visualization
import matplotlib.pyplot as plt

# Slightly advanced library for data visualization
import seaborn as sns

# A dictionary output that does not raise a key error
from collections import defaultdict

# A performance metrics in surprise
from surprise import accuracy

# Class is used to parse a file containing ratings, data should be in structure - user ; item ; rating
from surprise.reader import Reader

# Class for loading datasets
from surprise.dataset import Dataset

# For model tuning model hyper-parameters
from surprise.model_selection import GridSearchCV

# For splitting the rating data in train and test dataset
from surprise.model_selection import train_test_split

# For implementing similarity based recommendation system
from surprise.prediction_algorithms.knns import KNNBasic

# For implementing matrix factorization based recommendation system
from surprise.prediction_algorithms.matrix_factorization import SVD

# For implementing cross validation
from surprise.model_selection import KFold
```

Loading the data

```
In [13]: # Import the dataset
rating = pd.read_csv('ratings.csv')
#rating = pd.read_csv('/content/content/ratings.csv') # Uncomment this line code and comment above line of code
```

Let's check the **info** of the data

```
In [14]: rating.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100004 entries, 0 to 100003
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  -
0   userId      100004 non-null  int64
1   movieId     100004 non-null  int64
2   rating      100004 non-null  float64
3   timestamp   100004 non-null  int64
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
```

- There are **100,004 observations** and **4 columns** in the data
- All the columns are of **numeric data type**
- The data type of the timestamp column is int64 which is not correct. We can convert this to DateTime format but **we don't need timestamp for our analysis**. Hence, **we can drop this column**

```
In [15]: # Dropping timestamp column
rating = rating.drop(['timestamp'], axis=1)
```

Question 1: Exploring the dataset (7 Marks)

Let's explore the dataset and answer some basic data-related questions:

Q 1.1 Print the top 5 rows of the dataset (1 Mark)

```
In [17]: # Printing the top 5 rows of the dataset Hint: use .head()

# Remove _____ and complete the code
print(rating.head())
```

	userId	movieId	rating
0	1	31	2.5
1	1	1029	3.0
2	1	1061	3.0
3	1	1129	2.0
4	1	1172	4.0

```
In [21]: #Just for learning purposes

rating.sample(10)
```

```
Out[21]:
```

	userId	movieId	rating
3975	22	6333	4.5
49639	362	58559	5.0
74280	518	1088	4.0
75107	521	2100	3.0
95380	626	2395	2.0
57249	413	1073	4.0
59865	433	32	4.0
4676	23	25769	4.5
13846	90	733	4.0
55441	401	1198	5.0

```
In [22]: #Just for learning purposes

rating.tail()
```

```
Out[22]:
```

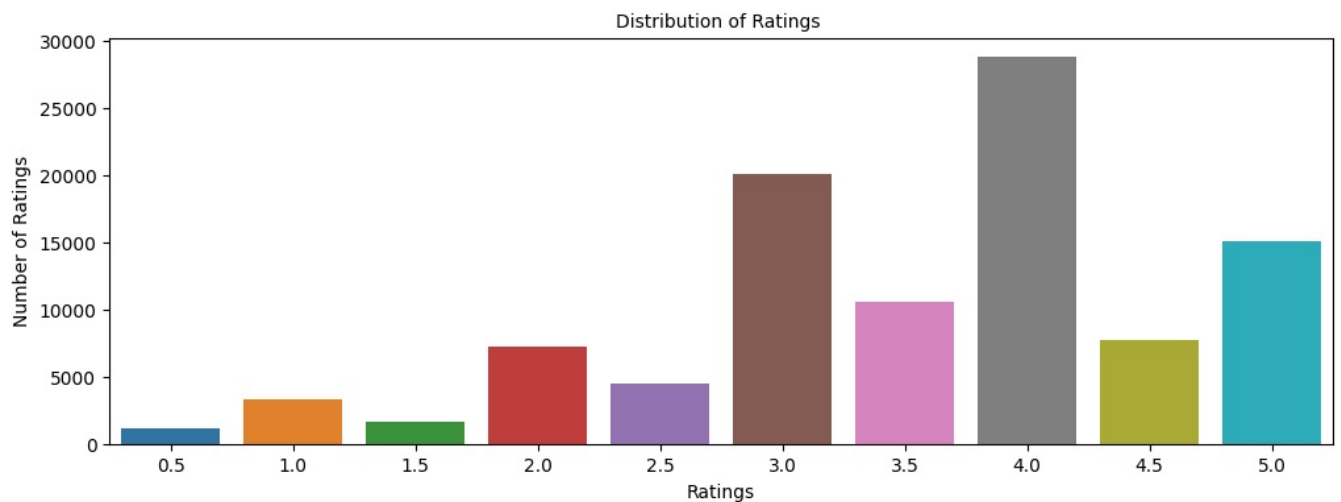
	userId	movieId	rating
99999	671	6268	2.5
100000	671	6269	4.0
100001	671	6365	4.0
100002	671	6385	2.5
100003	671	6565	3.5

Q 1.2 Describe the distribution of ratings. (1 Mark)

```
In [24]: plt.figure(figsize = (12, 4))
```

```
# Remove _____ and complete the code
sns.countplot(x="rating", data=rating)

plt.tick_params(labelsize = 10)
plt.title("Distribution of Ratings ", fontsize = 10)
plt.xlabel("Ratings", fontsize = 10)
plt.ylabel("Number of Ratings", fontsize = 10)
plt.show()
```



Write your Answer here:

The data seems to be **highly skewed** towards Rating "4", "3" and "5". Rating "4" is the most rated by the user, maybe due to a positive experience.

From 0.5 to 2.5 is possible that **users dont give ratings** because a bad movie experience. (movies they don't like)

Q 1.3 What is the total number of unique users and unique movies? (1 Mark)

```
In [25]: # Finding number of unique users
#remove _____ and complete the code
rating['userId'].nunique()
```

Out[25]: 671

Write your answer here:

Thera are **671** unique users in the dataset.

It means that the users have rated movies **more than one time**.

```
In [27]: # Finding number of unique movies
# Remove _____ and complete the code

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

Out[27]: 9066

Write your answer here:

This means that there are **movies that have got more than one Rating**.

Also it means that **not all user have rated all movies**.

The fact that not all users have rated all movies suggests that there may be some **missing data**, which could affect the accuracy of the recommendations generated by the system.

Q 1.4 Is there a movie in which the same user interacted with it more than once? (1 Mark)

```
In [29]: rating.groupby(['userId', 'movieId']).count()
```

```
Out[29]:
```

		rating
userId	movieId	
1	31	1
	1029	1
	1061	1
	1129	1
	1172	1
...
671	6268	1
	6269	1
	6365	1
	6385	1
	6565	1

100004 rows × 1 columns

```
In [31]: rating.groupby(['userId', 'movieId']).count()['rating'].sum()
Out[31]: 100004

In [34]: #For learning purposes
rating.groupby(['userId', 'movieId']).count()[rating.groupby(['userId', 'movieId']).count()['rating'] > 1].count()
Out[34]: 0
```

Write your Answer here:

There are no movies in which the same user interacted with it more than once

The user who hasn't seen the movie has not rated it.

Q 1.5 Which is the most interacted movie in the dataset? (1 Mark)

```
In [37]: # Remove _____ and complete the code
rating['movieId'].value_counts()
Out[37]:
```

356	341
296	324
318	311
593	304
260	291
...	...
98604	1
103659	1
104419	1
115927	1
6425	1

Name: movieId, Length: 9066, dtype: int64

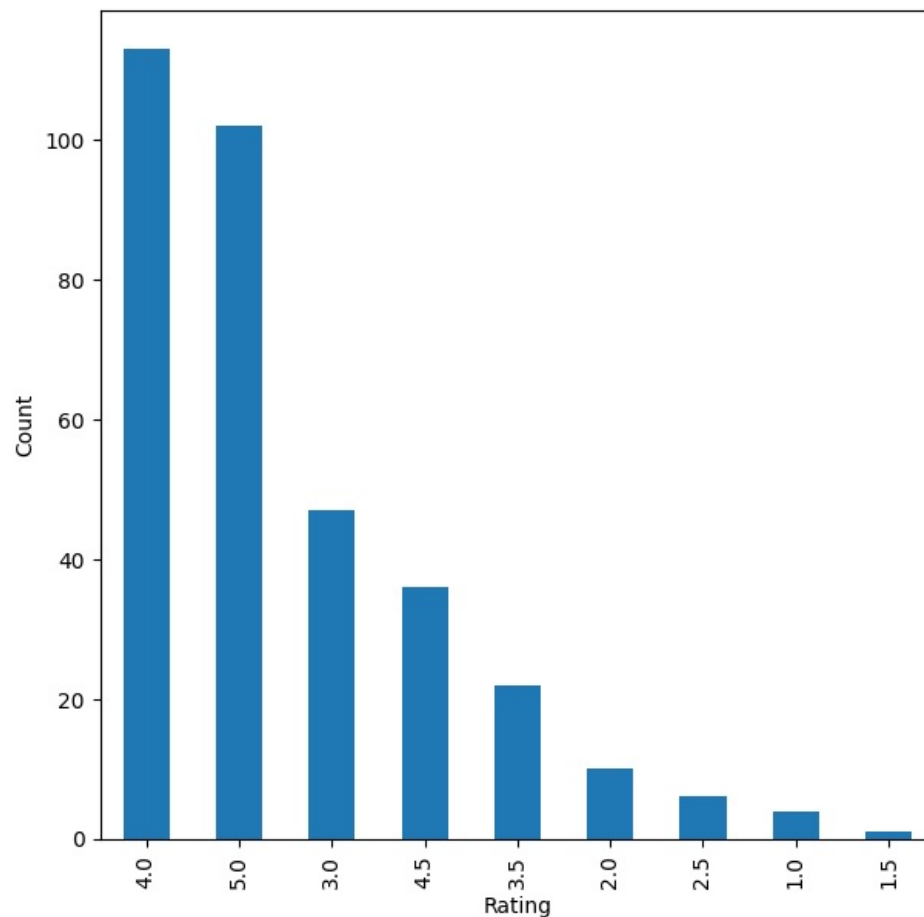
Write your Answer here:

The movie with **movieId 356 is the most interacted movie in the dataset** with 341 interactions. The second most interacted movie is with movieId 296 with 324 interactions.

```
In [39]: # Plotting distributions of ratings for 341 interactions with movieid 356
plt.figure(figsize=(7,7))

rating[rating['movieId'] == 356]['rating'].value_counts().plot(kind='bar')

plt.xlabel('Rating')
plt.ylabel('Count')
plt.show()
```



Write your Answer here:

The most interacted movie is actually **average**, there are a lot of people who did not like it.

Q 1.6 Which user interacted the most with any movie in the dataset? (1 Mark)

```
In [43]: # Remove _____ and complete the code
rating['userId'].value_counts()
```

```
Out[43]: 547    2391
564    1868
624    1735
15     1700
73     1610
...
296     20
289     20
249     20
221     20
1        20
Name: userId, Length: 671, dtype: int64
```

Write your Answer here:

The user with **ID 547** interacted the most with movies in the dataset.

Q 1.7 What is the distribution of the user-movie interactions in this dataset? (1 Mark)

```
In [47]: # Finding user-movie interactions distribution
count_interactions = rating.groupby('userId').count()['movieId']
count_interactions
```

```
Out[47]: userId
1         20
2         76
3         51
4        204
5        100
...
667        68
668        20
669        37
670        31
671       115
Name: movieId, Length: 671, dtype: int64
```

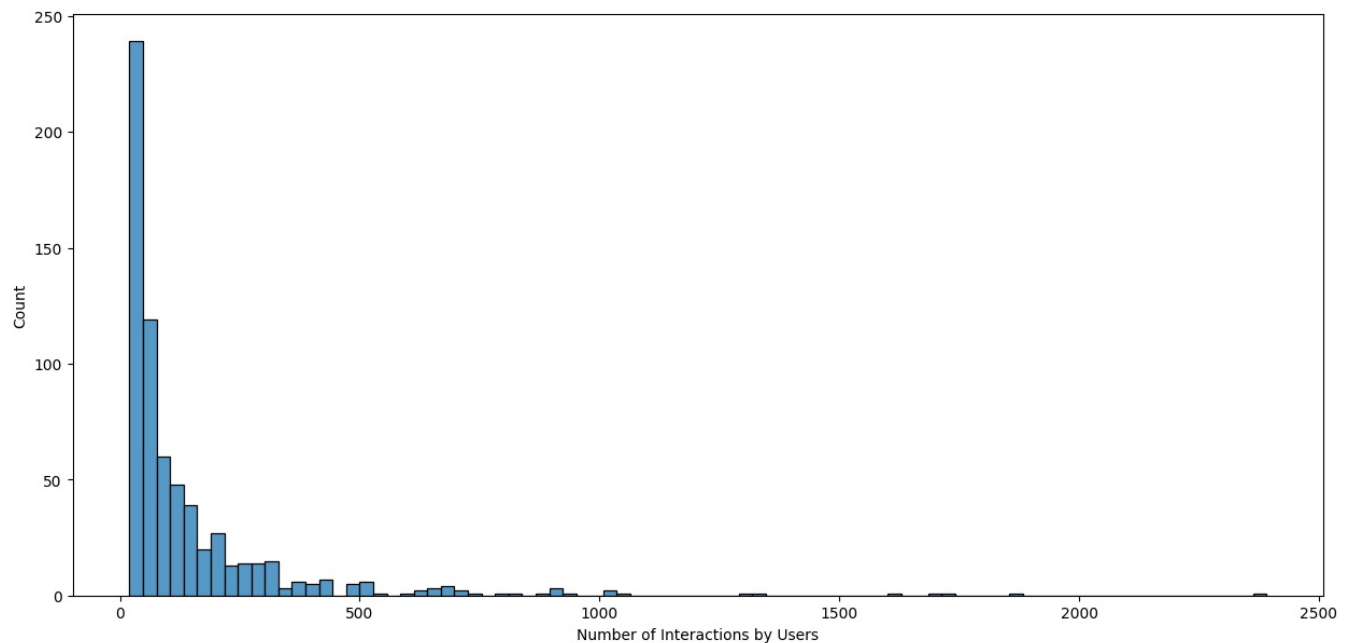
```
In [49]: # Plotting user-movie interactions distribution
```

```
plt.figure(figsize=(15,7))
# Remove _____ and complete the code

sns.histplot(count_interactions)

plt.xlabel('Number of Interactions by Users')

plt.show()
```



Write your Answer here:

The distribution of user-movie interactions in the dataset is **right-skewed**, which means that there are **more users who have interacted with a small number of movies, and very few users who have interacted with a large number of movies.**

This is evident from the histogram plot, which shows that the majority of users have **less than 200 interactions with movies, and very few users have more than 600 interactions.**

As we have now explored the data, let's start building Recommendation systems

Question 2: Create Rank-Based Recommendation System (3 Marks)

Model 1: Rank-Based Recommendation System

Rank-based recommendation systems provide recommendations based on the most popular items. This kind of recommendation system is useful when we have **cold start** problems. Cold start refers to the issue when we get a new user into the system and the machine is not able to recommend movies to the new user, as the user did not have any historical interactions in the dataset. In those cases, we can use rank-based recommendation system to recommend movies to the new user.

To build the rank-based recommendation system, we take **average** of all the ratings provided to each movie and then rank them based on their average rating.

```
In [50]: # Calculating average ratings
average_rating = rating.groupby('movieId')['rating'].mean()

# Calculating the count of ratings
count_rating = rating.groupby('movieId')['rating'].count()

# Making a dataframe with the count and average of ratings
final_rating = pd.DataFrame({'avg_rating': average_rating, 'rating_count': count_rating})
```

```
In [51]: final_rating.head()
```

```
Out[51]:
```

	avg_rating	rating_count
movieId		
1	3.872470	247
2	3.401869	107
3	3.161017	59
4	2.384615	13
5	3.267857	56

Now, let's create a function to find the **top n movies** for a recommendation based on the average ratings of movies. We can also add a **threshold for a minimum number of interactions** for a movie to be considered for recommendation.

```
In [52]: def top_n_movies(data, n, min_interaction=100):
#Finding movies with minimum number of interactions
recommendations = data[data['rating_count'] >= min_interaction]

#Sorting values w.r.t average rating
recommendations = recommendations.sort_values(by='avg_rating', ascending=False)

return recommendations.index[:n]
```

We can use this function with different n's and minimum interactions to get movies to recommend

Recommending top 5 movies with 50 minimum interactions based on popularity

```
In [55]: # Remove _____ and complete the code
list(top_n_movies(final_rating,5,50))
```

```
Out[55]: [858, 318, 969, 913, 1221]
```

Recommending top 5 movies with 100 minimum interactions based on popularity

```
In [56]: # Remove _____ and complete the code
list(top_n_movies(final_rating,5,100))
```

```
Out[56]: [858, 318, 1221, 50, 527]
```

Recommending top 5 movies with 200 minimum interactions based on popularity

```
In [57]: # Remove _____ and complete the code
list(top_n_movies(final_rating,5,200))
```

```
Out[57]: [858, 318, 50, 527, 608]
```

Now that we have seen **how to apply the Rank-Based Recommendation System**, let's apply the **Collaborative Filtering Based Recommendation Systems**.

Model 2: User based Collaborative Filtering Recommendation System (7 Marks)

Users	Movies					
	Forrest Gump	Cast Away	Captain Philips	The Terminal	The Terminator	The Matrix
A	1	1	1	1	0	0
B	1	1	1	?	0	0
C	0	0	0	?	1	1

In the above **interactions matrix**, out of users B and C, which user is most likely to interact with the movie, "The Terminal"?

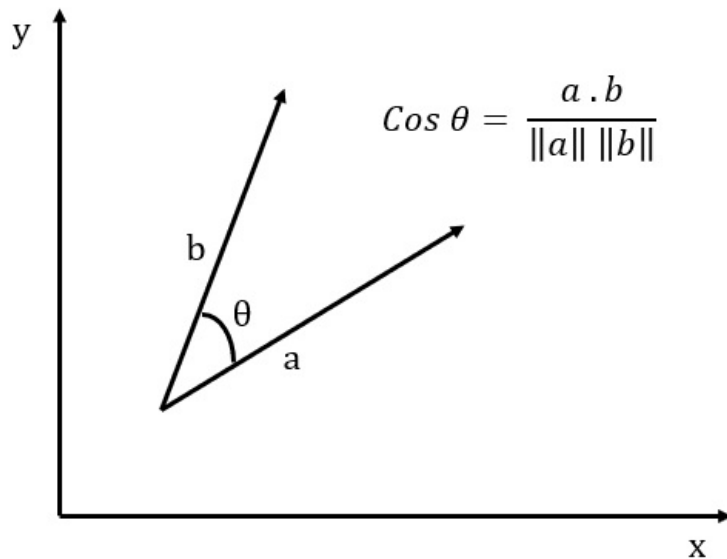
In this type of recommendation system, **we do not need any information** about the users or items. We only need user item interaction data to build a collaborative recommendation system. For example -

1. **Ratings** provided by users. For example - ratings of books on goodread, movie ratings on imdb etc
2. **Likes** of users on different facebook posts, likes on youtube videos
3. **Use/buying** of a product by users. For example - buying different items on e-commerce sites
4. **Reading** of articles by readers on various blogs

Types of Collaborative Filtering

- Similarity/Neighborhood based
 - User-User Similarity Based
 - Item-Item similarity based
- Model based

Building Similarity/Neighborhood based Collaborative Filtering



Building a baseline user-user similarity based recommendation system

- Below, we are building **similarity-based recommendation systems** using **cosine** similarity and using **KNN to find similar users** which are the nearest neighbor to the given user.
- We will be using a new library, called **surprise**, to build the remaining models. Let's first import the necessary classes and functions from this library.

Below we are loading the **rating** dataset, which is a **pandas DataFrame**, into a **different format called surprise.dataset.DatasetAutoFolds**, which is required by this library. To do this, we will be using the classes **Reader** and **Dataset**. Finally splitting the data into train and test set.

Making the dataset into surprise dataset and splitting it into train and test set

```
In [58]: # Instantiating Reader scale with expected rating scale
reader = Reader(rating_scale=(0, 5))

# Loading the rating dataset
data = Dataset.load_from_df(rating[['userId', 'movieId', 'rating']], reader)

# Splitting the data into train and test dataset
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)
```

Build the first baseline similarity based recommendation system using cosine similarity and KNN

```
In [59]: # Remove _____ and complete the code

# Defining Nearest neighbour algorithm
algo_knn_user = KNNBasic(sim_options={'user_based': True})

# Train the algorithm on the trainset or fitting the model on train dataset
algo_knn_user.fit(trainset)

# Predict ratings for the testset
predictions = algo_knn_user.test(testset)

# Then compute RMSE
accuracy.rmse(predictions)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.9672
0.9671524784958385
```

Out[59]:

Q 3.1 What is the RMSE for baseline user based collaborative filtering recommendation

system? (1 Mark)

Write your Answer here:

The RMSE for the baseline user-based collaborative filtering recommendation system is 0.9672. It is **highly accurate on the 0 to 5 range**.

Q 3.2 What is the Predicted rating for an user with userId=4 and for movieId=10 and movieId=3? (1 Mark)

Let's us now predict rating for an user with `userId=4` and for `movieId=10`

```
In [60]: # Remove _____ and complete the code
algo_knn_user.predict(4, 10, r_ui=4, verbose=True)

user: 4          item: 10          r_ui = 4.00    est = 3.68    {'actual_k': 40, 'was_impossible': False}
Out[60]: Prediction(uid=4, iid=10, r_ui=4, est=3.680292946370331, details={'actual_k': 40, 'was_impossible': False})
```

Write your Answer here:

The predicted rating for user with `userId=4` and `movieId=10` is 3.68.

Let's predict the rating for the same `userId=4` but for a movie which this user has not interacted before i.e. `movieId=3`

```
In [61]: # Remove _____ and complete the code
algo_knn_user.predict(4, 3, r_ui=4, verbose=True)

user: 4          item: 3          r_ui = 4.00    est = 3.31    {'actual_k': 40, 'was_impossible': False}
Out[61]: Prediction(uid=4, iid=3, r_ui=4, est=3.3125531823311247, details={'actual_k': 40, 'was_impossible': False})
```

Write your Answer here:

The predicted rating for user with `userId=4` and `movieId=3` is 3.31.

Improving user-user similarity based recommendation system by tuning its hyper-parameters

Below we will be tuning hyper-parameters for the `KNNBasic` algorithms. Let's try to understand different hyperparameters of `KNNBasic` algorithm -

- **k** (int) – The (max) number of neighbors to take into account for aggregation (see this note). Default is 40.
- **min_k** (int) – The minimum number of neighbors to take into account for aggregation. If there are not enough neighbors, the prediction is set to the global mean of all ratings. Default is 1.
- **sim_options** (dict) – A dictionary of options for the similarity measure. And there are four similarity measures available in surprise -
 - cosine
 - msd (default)
 - pearson
 - pearson baseline

For more details please refer the official documentation https://surprise.readthedocs.io/en/stable/knn_inspired.html

Q 3.3 Perform hyperparameter tuning for the baseline user based collaborative filtering recommendation system and find the RMSE for tuned user based collaborative filtering recommendation system? (3 Marks)

```
In [62]: # Remove _____ and complete the code

# Setting up parameter grid to tune the hyperparameters
param_grid = {'k': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100], 'sim_options': {'name': ['msd', 'cosine', 'pearson']

# Performing 3-fold cross validation to tune the hyperparameters
grid_obj = GridSearchCV(KNNBasic, param_grid, measures=['rmse', 'mae'], cv=3, n_jobs=-1)

# Fitting the data
grid_obj.fit(data)

# Best RMSE score
print(grid_obj.best_score['rmse'])

# Combination of parameters that gave the best RMSE score
print(grid_obj.best_params['rmse'])

0.9773631221515373
{'k': 20, 'sim_options': {'name': 'msd', 'user_based': True}}
```

Once the grid search is **complete**, we can get the **optimal values for each of those hyperparameters** as shown above.

Below we are analysing evaluation metrics - RMSE and MAE at each and every split to analyze the impact of each value of hyperparameters

```
In [63]: results_df = pd.DataFrame.from_dict(grid_obj.cv_results)
results_df.head()
```

```
Out[63]:
```

	split0_test_rmse	split1_test_rmse	split2_test_rmse	mean_test_rmse	std_test_rmse	rank_test_rmse	split0_test_mae	split1_test_mae	split
0	0.983374	0.977378	0.981472	0.980741	0.002502	3	0.753261	0.748600	
1	1.022508	1.016765	1.017815	1.019029	0.002497	29	0.789230	0.784350	
2	1.026503	1.024715	1.025937	1.025718	0.000746	30	0.794253	0.794393	
3	0.979831	0.974810	0.977448	0.977363	0.002051	1	0.751467	0.747287	
4	1.009071	1.004716	1.005485	1.006424	0.001898	19	0.778311	0.774799	

Now, let's build the **final model by using tuned values of the hyperparameters**, which we received by using **grid search cross-validation**.

```
In [64]: # Remove _____ and complete the code

# Using the optimal similarity measure for user-user based collaborative filtering
# Creating an instance of KNNBasic with optimal hyperparameter values
similarity_algo_optimized_user = KNNBasic(k=20, sim_options={'name': 'msd', 'user_based': True}, verbose=False)

# Training the algorithm on the trainset
similarity_algo_optimized_user.fit(trainset)

# Predicting ratings for the testset
predictions = similarity_algo_optimized_user.test(testset)

# Computing RMSE on testset
accuracy.rmse(predictions)
```

```
RMSE: 0.9625
0.9625200448463395
```

```
Out[64]:
```

Write your Answer here:

The RMSE of the **optimized** user-based collaborative filtering model is 0.9625.

Q 3.4 What is the Predicted rating for an user with userId =4 and for movieId= 10 and movieId=3 using tuned user based collaborative filtering? (1 Mark)

Let's us now predict rating for an user with `userId=4` and for `movieId=10` with the optimized model

```
In [65]: # Remove _____ and complete the code
similarity_algo_optimized_user.predict(4, 10, r_ui=4, verbose=True)

user: 4      item: 10      r_ui = 4.00      est = 3.74      {'actual_k': 20, 'was_impossible': False}
Out[65]: Prediction(uid=4, iid=10, r_ui=4, est=3.740028692988536, details={'actual_k': 20, 'was_impossible': False})
```

Write your Answer here:

The predicted rating for user with userId=4 and movieId=10 using the optimized user-based collaborative filtering model is **3.74**.

Below we are predicting rating for the same `userId=4` but for a movie which this user has not interacted before i.e. `movieId=3`, by using the optimized model as shown below -

```
In [66]: # Remove _____ and complete the code
similarity_algo_optimized_user.predict(4, 3, r_ui=4, verbose=True)

user: 4      item: 3      r_ui = 4.00      est = 3.72      {'actual_k': 20, 'was_impossible': False}
```

```
Prediction(uid=4, iid=3, r_ui=4, est=3.7228745701935386, details={'actual_k': 20, 'was_impossible': False})
```

Out[66]:

Write your Answer here:

The predicted rating for user with userId=4 and movieId=3 using the optimized user-based collaborative filtering model is **3.72**.

Identifying similar users to a given user (nearest neighbors)

We can also find out the similar users to a given user or its nearest neighbors based on this KNNBasic algorithm. Below we are finding 5 most similar user to the `userId=4` based on the `msd` distance metric

```
In [67]: similarity_algo_optimized_user.get_neighbors(4, k=5)
```

Out[67]: [665, 417, 647, 654, 260]

Implementing the recommendation algorithm based on optimized KNNBasic model

Below we will be implementing a function where the input parameters are -

- data: a rating dataset
- user_id: an user id against which we want the recommendations
- top_n: the number of movies we want to recommend
- algo: the algorithm we want to use to predict the ratings

```
In [68]: def get_recommendations(data, user_id, top_n, algo):  
  
    # Creating an empty list to store the recommended movie ids  
    recommendations = []  
  
    # Creating an user item interactions matrix  
    user_item_interactions_matrix = data.pivot(index='userId', columns='movieId', values='rating')  
  
    # Extracting those movie ids which the user_id has not interacted yet  
    non_interacted_movies = user_item_interactions_matrix.loc[user_id][user_item_interactions_matrix.loc[user_id].isna().index]  
  
    # Looping through each of the movie id which user_id has not interacted yet  
    for item_id in non_interacted_movies:  
  
        # Predicting the ratings for those non interacted movie ids by this user  
        est = algo.predict(user_id, item_id).est  
  
        # Appending the predicted ratings  
        recommendations.append((item_id, est))  
  
    # Sorting the predicted ratings in descending order  
    recommendations.sort(key=lambda x: x[1], reverse=True)  
  
    return recommendations[:top_n] # returning top n highest predicted rating movies for this user
```

Predicted top 5 movies for userId=4 with similarity based recommendation system

```
In [74]: #remove _____ and complete the code  
user_id = 4  
top_n = 5  
recommendations = get_recommendations(rating, user_id, top_n, similarity_algo_optimized_user)
```

Q 3.5 Predict the top 5 movies for userId=4 with similarity based recommendation system (1 Mark)

```
In [77]: recommendations
```

Out[77]: [(53, 5), (183, 5), (301, 5), (309, 5), (559, 5)]

Model 3: Item based Collaborative Filtering Recommendation System (7 Marks)

```
In [78]: # Remove _____ and complete the code  
  
# Defining similarity measure  
sim_options = {'user_based': False}  
  
# Defining Nearest neighbour algorithm  
algo_knn_item = KNNBasic(sim_options=sim_options, verbose=False)  
  
# Train the algorithm on the trainset or fitting the model on train dataset
```

```

algo_knn_item.fit(trainset)

# Predict ratings for the testset
predictions = algo_knn_item.test(testset)

# Then compute RMSE
accuracy.rmse(predictions)

```

RMSE: 0.9433
0.9433248796999978

Out[78]:

Q 4.1 What is the RMSE for baseline item based collaborative filtering recommendation system ?(1 Mark)

Write your Answer here:

The RMSE for baseline item-based collaborative filtering recommendation system is **0.9433**.

Let's us now predict rating for an user with `userId=4` and for `movieId=10`

Q 4.2 What is the Predicted rating for an user with `userId =4` and for `movieId= 10` and `movieId=3`? (1 Mark)

In [79]: `# Remove _____ and complete the code`
`algo_knn_item.predict(4, 10, r_ui=4, verbose=True)`

user: 4 item: 10 r_ui = 4.00 est = 4.26 {'actual_k': 40, 'was_impossible': False}
 Out[79]: Prediction(uid=4, iid=10, r_ui=4, est=4.255054787154994, details={'actual_k': 40, 'was_impossible': False})

Write your Answer here:

For user with `userId=4` and `movieId=10`, the predicted rating using baseline item-based collaborative filtering recommendation system is **4.26**.

Let's predict the rating for the same `userId=4` but for a movie which this user has not interacted before i.e. `movieId=3`

In [80]: `# Remove _____ and complete the code`
`algo_knn_item.predict(4, 3, r_ui=4, verbose=True)`

user: 4 item: 3 r_ui = 4.00 est = 3.87 {'actual_k': 40, 'was_impossible': False}
 Out[80]: Prediction(uid=4, iid=3, r_ui=4, est=3.865175609312417, details={'actual_k': 40, 'was_impossible': False})

Write your Answer here:

The predicted rating for an user with `userId=4` and for `movieId=3` using the optimized item-based collaborative filtering model is **3.87**.

Q 4.3 Perform hyperparameter tuning for the baseline item based collaborative filtering recommendation system and find the RMSE for tuned item based collaborative filtering recommendation system? (3 Marks)

In [81]: `# Remove _____ and complete the code`

```

# Setting up parameter grid to tune the hyperparameters
param_grid = {'k': [10, 20, 30, 40, 50],
              'sim_options': {'name': ['cosine', 'pearson', 'msd'],
                              'user_based': [False]}}

# Performing 3-fold cross validation to tune the hyperparameters
grid_obj = GridSearchCV(KNNBasic, param_grid, measures=['rmse', 'mae'], cv=3, n_jobs=-1)

# Fitting the data
grid_obj.fit(data)

# Best RMSE score
print(grid_obj.best_score['rmse'])

# Combination of parameters that gave the best RMSE score
print(grid_obj.best_params['rmse'])

```

0.9394418338338363
 {'k': 50, 'sim_options': {'name': 'msd', 'user_based': False}}

Once the **grid search** is complete, we can get the **optimal values for each of those hyperparameters as shown above**

Below we are analysing evaluation metrics - RMSE and MAE at each and every split to analyze the impact of each value of hyperparameters

In [82]: `results_df = pd.DataFrame.from_dict(grid_obj.cv_results)`

```
results_df.head()
```

```
Out[82]:
```

	split0_test_rmse	split1_test_rmse	split2_test_rmse	mean_test_rmse	std_test_rmse	rank_test_rmse	split0_test_mae	split1_test_mae	split
0	1.045124	1.048939	1.044085	1.046049	0.002087	15	0.817022	0.816530	
1	1.031477	1.032418	1.031839	1.031912	0.000387	14	0.802078	0.801146	
2	0.973482	0.977705	0.979943	0.977043	0.002679	5	0.752172	0.753793	
3	1.011446	1.017831	1.015332	1.014870	0.002627	13	0.789249	0.791205	
4	1.007417	1.009542	1.008070	1.008343	0.000889	12	0.780655	0.781957	

Now let's build the **final model** by using **tuned values of the hyperparameters** which we received by using grid search cross-validation.

```
In [87]: # Remove _____ and complete the code
# Creating an instance of KNNBasic with optimal hyperparameter values
similarity_algo_optimized_item = KNNBasic(sim_options={'name': 'pearson_baseline', 'user_based': False}, k=50,

# Training the algorithm on the trainset
similarity_algo_optimized_item.fit(trainset)

# Predicting ratings for the testset
predictions = similarity_algo_optimized_item.test(testset)

# Computing RMSE on testset
print("RMSE for tuned item-based collaborative filtering: {}".format(accuracy.rmse(predictions)))
```

RMSE: 0.9484

RMSE for tuned item-based collaborative filtering: 0.9484073266816083

Write your Answer here:

The difference between the two RMSE values is quite small, indicating that **both models are performing similarly well**.

Based on these results, we can conclude that the tuned item-based collaborative filtering model with the hyperparameters $k=50$ and similarity measure of 'msd' is a **good choice for making movie recommendations to users**.

Q 4.4 What is the Predicted rating for an item with $userId=4$ and for $movieId=10$ and $movieId=3$ using tuned item based collaborative filtering? (1 Mark)

Let's us now predict rating for an user with $userId=4$ and for $movieId=10$ with the optimized model as shown below

```
In [100]: # Remove _____ and complete the code
# Creating an instance of KNNBasic with tuned hyperparameter values
similarity_algo_optimized_item.predict(4, 10, r_ui=4, verbose=True)

user: 4      item: 10      r_ui = 4.00  est = 4.31  {'actual_k': 50, 'was_impossible': False}
Out[100]: Prediction(uid=4, iid=10, r_ui=4, est=4.313881303017231, details={'actual_k': 50, 'was_impossible': False})
```

Write your Answer here:

The predicted rating for an item with $userId=4$ and for $movieId=10$ using tuned item-based collaborative filtering is **4.31**.

Let's predict the rating for the same $userId=4$ but for a movie which this user has not interacted before i.e. $movieId=3$, by using the optimized model:

```
In [101]: # Remove _____ and complete the code
similarity_algo_optimized_item.predict(4, 3, verbose=True)

user: 4      item: 3      r_ui = None  est = 3.87  {'actual_k': 50, 'was_impossible': False}
Out[101]: Prediction(uid=4, iid=3, r_ui=None, est=3.8679817054210965, details={'actual_k': 50, 'was_impossible': False})
```

Write your Answer here:

The predicted rating for user with $userId=4$ and $movieId=3$ using the optimized item-based collaborative filtering model is **3.87**.

Identifying similar items to a given item (nearest neighbors)

Identifying similar items to a given item (nearest neighbors)

We can also find out the similar items to a given item or its nearest neighbors based on this KNNBasic algorithm. Below we are finding 5 most similar items to the `movieId=3` based on the `msd` distance metric

```
In [102]: # Remove _____ and complete the code
similarity_algo_optimized_item.get_neighbors(3, k=5)

Out[102]: [31, 37, 42, 48, 73]
```

Predicted top 5 movies for userId=4 with similarity based recommendation system

```
In [103]: # Remove _____ and complete the code
recommendations = get_recommendations(rating, 4, 5, similarity_algo_optimized_item)
```

Q 4.5 Predict the top 5 movies for userId=4 with similarity based recommendation system (1 Mark)

```
In [104]: recommendations

Out[104]: [(84, 5), (1040, 5), (2481, 5), (2562, 5), (3078, 5)]
```

Model 4: Based Collaborative Filtering - Matrix Factorization using SVD (7 Marks)

Model-based Collaborative Filtering is a **personalized recommendation system**, the recommendations are based on the past behavior of the user and it is not dependent on any additional information. We use **latent features** to find recommendations for each user.

Latent Features: The features that are not present in the empirical data but can be inferred from the data. For example:

User	Movie	Rating
Ram	Rambo	8
Hari	The Notebook	7
Rahi	The Usual Suspects	8
Rahim	Dumb and Dumber	7

Now if we notice the above movies closely:

	Action	Romance	Suspense	Comedy
Rambo	Yes	No	No	No
The Notebook	No	Yes	No	No
The Usual Suspects	Yes	No	Yes	No
Dumb and Dumber	No	No	No	Yes

Here **Action**, **Romance**, **Suspense** and **Comedy** are latent features of the corresponding movies. Similarly, we can compute the latent features for users as shown below:

	Action	Romance	Suspense	Comedy
Ram	Yes	No	No	No
Hari	No	Yes	No	No
Rahi	Yes	No	Yes	No
Rahim	No	No	No	Yes

Singular Value Decomposition (SVD)

SVD is used to **compute the latent features** from the **user-item matrix**. But SVD does not work when we **miss values** in the **user-item matrix**.

First we need to convert the below movie-rating dataset:

User	Movie	Rating
Ram	Rambo	8
Hari	The Notebook	7
Rahi	The Usual Suspects	8

Rahim	Dumb and Dumber	7
-------	-----------------	---

into an user-item matrix as shown below:

	Rambo	The Notebook	The Usual Suspects	Dumb and Dumber
Ram	8	0	0	0
Hari	0	7	0	0
Rahi	0	0	8	0
Rahim	0	0	0	7

We have already done this above while computing cosine similarities.

SVD decomposes this above matrix into three separate matrices:

- U matrix
- Sigma matrix
- V transpose matrix

U-matrix

	Action	Romance	Suspense	Comedy
Ram	8	0	2	0
Hari	0	9	1	2
Rahi	7	1	8	1
Rahim	0	2	0	8

the above matrix is a $n \times k$ matrix, where:

- n is number of users
- k is number of latent features

Sigma-matrix

	Action	Romance	Suspense	Comedy
Action	20	0	0	0
Romance	0	15	0	0
Suspense	0	0	30	0
Comedy	0	0	0	25

the above matrix is a $k \times k$ matrix, where:

- k is number of latent features
- Each diagonal entry is the singular value of the original interaction matrix

V-transpose matrix

	Rambo	The Notebook	The Usual Suspects	Dumb and Dumber
Action	8	0	5	0
Romance	0	7	2	5
Suspense	2	0	8	0
Comedy	0	1	1	8

the above matrix is a $k \times n$ matrix, where:

- k is the number of latent features
- n is the number of items

Build a baseline matrix factorization recommendation system

```
In [105_ # Remove _____ and complete the code
# Using SVD matrix factorization
```



```

algo_svd = SVD()

# Training the algorithm on the trainset
algo_svd.fit(trainset)

# Predicting ratings for the testset
predictions = algo_svd.test(testset)

# Computing RMSE on the testset
accuracy.rmse(predictions)

```

RMSE: 0.9046
0.9045593128225919

Out[105]:

Q 5.1 What is the RMSE for baseline SVD based collaborative filtering recommendation system? (1 Mark)

Write your Answer here:

The RMSE for the baseline SVD-based collaborative filtering recommendation system is 0.9046.

Q 5.2 What is the Predicted rating for an user with userId =4 and for movieId= 10 and movieId=3? (1 Mark)

Let's us now predict rating for an user with `userId=4` and for `movieId=10`

In [106]..

```

# Remove _____ and complete the code
algo_svd.predict(4, 10, r_ui=4, verbose=True)

```

user: 4 item: 10 r_ui = 4.00 est = 3.97 {'was_impossible': False}
Prediction(uid=4, iid=10, r_ui=4, est=3.972208830134703, details={'was_impossible': False})

Out[106]:

Write your Answer here:

The userId = 4 and movieId = 10, the predicted rating using SVD-based collaborative filtering is 3.97.

Let's predict the rating for the same `userId=4` but for a movie which this user has not interacted before i.e. `movieId=3` :

In [107]..

```

# Remove _____ and complete the code
algo_svd.predict(4, 3, verbose=True)

```

user: 4 item: 3 r_ui = None est = 3.85 {'was_impossible': False}
Prediction(uid=4, iid=3, r_ui=None, est=3.848019385882973, details={'was_impossible': False})

Out[107]:

Write your Answer here:

The userId = 4 and movieId = 3, the predicted rating using SVD-based collaborative filtering is 3.85.

Improving matrix factorization based recommendation system by tuning its hyper-parameters

In SVD, rating is predicted as -

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u$$

If user u

u

is unknown, then the bias b_u

b_u

and the factors p_u

p_u

are assumed to be zero. The same applies for item i

i

with b_i

b_i

and q_i

q_i

.

To estimate all the unknown, we minimize the following regularized squared error:

$$\sum_{r_{ui} \in R_{\text{train}}} (r_{ui} - \hat{r}_{ui})^2 + \lambda (b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2)$$

The minimization is performed by a very straightforward **stochastic gradient descent**:

$$\begin{aligned}b_u &\leftarrow b_u + \gamma(e_{ui} - \lambda b_u) \\b_i &\leftarrow b_i + \gamma(e_{ui} - \lambda b_i) \\p_u &\leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda p_u) \\q_i &\leftarrow q_i + \gamma(e_{ui} \cdot p_u - \lambda q_i)\end{aligned}$$

There are many hyperparameters to tune in this algorithm, you can find a full list of hyperparameters [here](#)

Below we will be tuning only three hyperparameters -

- **n_epochs**: The number of iteration of the SGD algorithm
- **lr_all**: The learning rate for all parameters
- **reg_all**: The regularization term for all parameters

Q 5.3 Perform hyperparameter tuning for the baseline SVD based collaborative filtering recommendation system and find the RMSE for tuned SVD based collaborative filtering recommendation system? (3 Marks)

```
In [108.. # Remove _____ and complete the code

# Set the parameter space to tune
param_grid = {'n_epochs': [10, 20, 30], 'lr_all': [0.001, 0.005, 0.01],
              'reg_all': [0.2, 0.4, 0.6]}

# Performing 3-fold gridsearch cross validation
gs = GridSearchCV(SVD, param_grid, measures=['rmse', 'mae'], cv=3, n_jobs=-1)

# Fitting data
gs.fit(data)

# Best RMSE score
print(gs.best_score['rmse'])

# Combination of parameters that gave the best RMSE score
print(gs.best_params['rmse'])

0.8940344791064283
{'n_epochs': 30, 'lr_all': 0.01, 'reg_all': 0.2}
```

Once the **grid search** is complete, we can get the **optimal values** for each of those hyperparameters, as shown above.

Below we are analysing evaluation metrics - RMSE and MAE at each and every split to analyze the impact of each value of hyperparameters

```
In [109.. results_df = pd.DataFrame.from_dict(gs.cv_results)
results_df.head()
```

```
Out[109]:
```

	split0_test_rmse	split1_test_rmse	split2_test_rmse	mean_test_rmse	std_test_rmse	rank_test_rmse	split0_test_mae	split1_test_mae	split2_test_mae	mean_test_mae
0	0.937830	0.944753	0.947629	0.943404	0.004113	25	0.732986	0.738467	0.743048	0.738184
1	0.942400	0.948925	0.952211	0.947845	0.004077	26	0.738383	0.743048	0.748710	0.743414
2	0.946978	0.954066	0.957849	0.952964	0.004506	27	0.743789	0.748710	0.753421	0.748640
3	0.902877	0.907902	0.911292	0.907357	0.003457	11	0.698676	0.701458	0.704239	0.701458
4	0.909389	0.914692	0.919014	0.914365	0.003936	15	0.705838	0.708895	0.711952	0.708578

Now, we will **the build final model** by using **tuned values** of the hyperparameters, which we received using grid search cross-validation above.

```
In [110]: # Remove _____ and complete the code

# Building the optimized SVD model using optimal hyperparameter search
svd_algo_optimized = SVD(n_epochs=30, lr_all=0.01, reg_all=0.2)

# Training the algorithm on the trainset
svd_algo_optimized.fit(trainset)

# Predicting ratings for the testset
predictions = svd_algo_optimized.test(testset)

# Computing RMSE
accuracy.rmse(predictions)
```

```
RMSE: 0.8956
0.8955806966443421
```

Out[110]:

Q 5.4 What is the Predicted rating for an user with userId =4 and for movieId= 10 and movieId=3 using SVD based collaborative filtering? (1 Mark)

Let's us now predict rating for an user with `userId=4` and for `movieId=10` with the optimized model

```
In [111]: # Remove _____ and complete the code
svd_algo_optimized.predict(4, 10, r_ui=4, verbose=True)

user: 4      item: 10      r_ui = 4.00  est = 3.99  {'was_impossible': False}
Out[111]: Prediction(uid=4, iid=10, r_ui=4, est=3.988826318072218, details={'was_impossible': False})
```

Write your Answer here:

The predicted rating for user 4 and movie 10 using the optimized SVD-based collaborative filtering model is 3.99, which is slightly better than the previous SVD model's prediction of 3.97.

Let's predict the rating for the same `userId=4` but for a movie which this user has not interacted before i.e. `movieId=3` :

```
In [112]: # Remove _____ and complete the code
svd_algo_optimized.predict(4, 3, verbose=True)

user: 4      item: 3      r_ui = None  est = 3.65  {'was_impossible': False}
Out[112]: Prediction(uid=4, iid=3, r_ui=None, est=3.6469962362579333, details={'was_impossible': False})
```

Q 5.5 Predict the top 5 movies for userId=4 with SVD based recommendation system?(1 Mark)

```
In [115]: # Remove _____ and complete the code
user_id = 4
top_n = 5
recommendations = get_recommendations(rating, user_id, top_n, svd_algo_optimized)
print(recommendations)

[(116, 4.967444093465211), (1192, 4.958558923310242), (926, 4.940267693925108), (5114, 4.928352807158776), (1948, 4.921675138644459)]
```

Predicting ratings for already interacted movies

Below we are comparing the rating predictions of users for those movies which has been already watched by an user. This will help us to understand how well are predictions are as compared to the actual ratings provided by users

```
In [116]: def predict_already_interacted_ratings(data, user_id, algo):

    # Creating an empty list to store the recommended movie ids
    recommendations = []

    # Creating an user item interactions matrix
    user_item_interactions_matrix = data.pivot(index='userId', columns='movieId', values='rating')

    # Extracting those movie ids which the user_id has interacted already
    interacted_movies = user_item_interactions_matrix.loc[user_id][user_item_interactions_matrix.loc[user_id].n

    # Looping through each of the movie id which user_id has interacted already
    for item_id in interacted_movies:

        # Extracting actual ratings
        actual_rating = user_item_interactions_matrix.loc[user_id, item_id]

        # Predicting the ratings for those non interacted movie ids by this user
```

```

predicted_rating = algo.predict(user_id, item_id).est

# Appending the predicted ratings
recommendations.append((item_id, actual_rating, predicted_rating))

# Sorting the predicted ratings in descending order
recommendations.sort(key=lambda x: x[1], reverse=True)

return pd.DataFrame(recommendations, columns=['movieId', 'actual_rating', 'predicted_rating']) # returning t

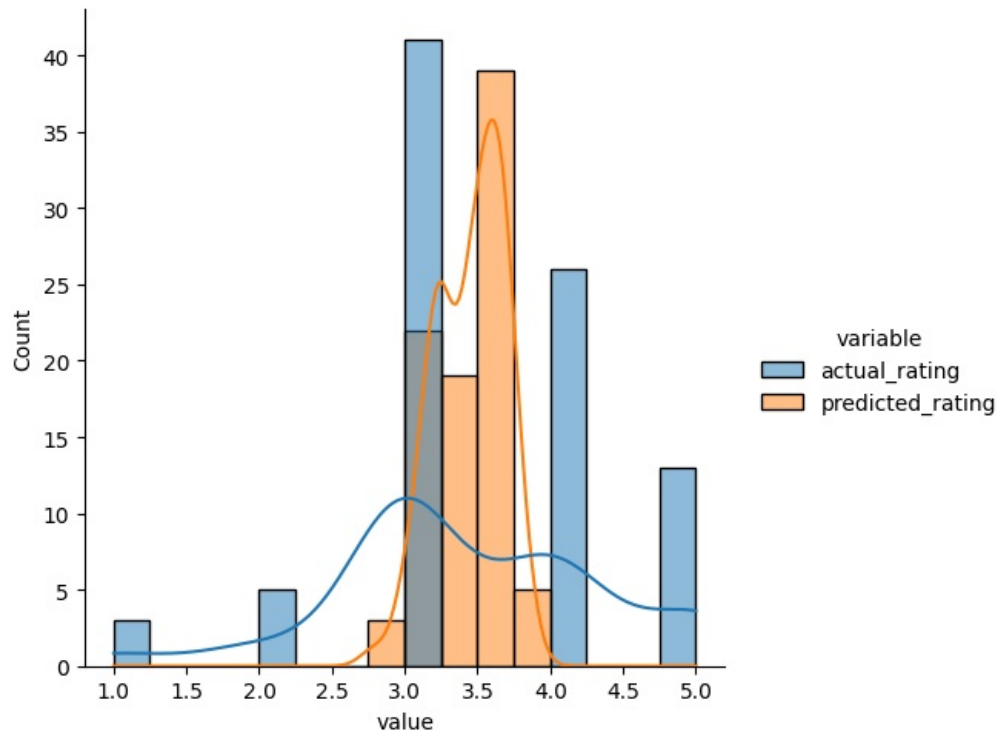
```

Here we are comparing the predicted ratings by `similarity based recommendation` system against actual ratings for `userId=7`

```

In [127.. predicted_ratings_for_interacted_movies = predict_already_interacted_ratings(rating, 7, similarity_algo_optimiz
df = predicted_ratings_for_interacted_movies.melt(id_vars='movieId', value_vars=['actual_rating', 'predicted_ra
sns.displot(data=df, x='value', hue='variable', kde=True);

```



Write your Answer here:

After analyzing the graphs and comparing the results with other userIDs, it can be observed that there is a significant difference between the predicted and actual ratings in most cases.

However, there are instances where the predicted ratings are quite close to the actual ones. This indicates that the model's accuracy may not be tuned correctly.

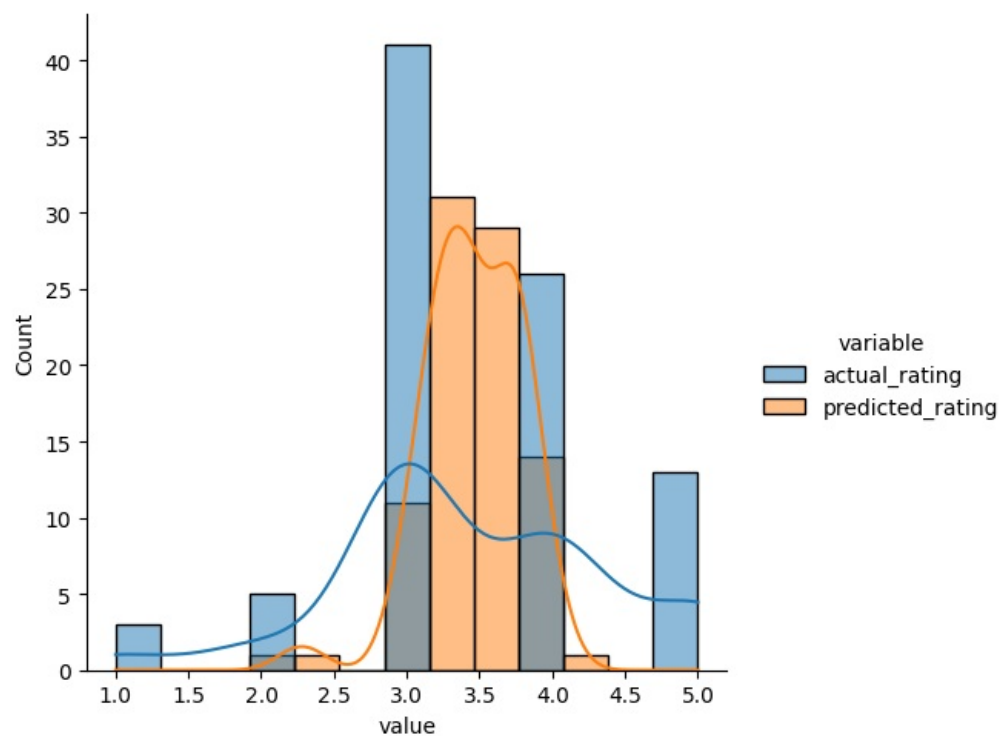
Nonetheless, it's essential to note that this inaccuracy could also be due to the **users different genre preferences**, which are not accounted for in the model.

Below we are comparing the predicted ratings by `matrix factorization based recommendation` system against actual ratings for `userId=7`

```

In [118.. predicted_ratings_for_interacted_movies = predict_already_interacted_ratings(rating, 7, svd_algo_optimized)
df = predicted_ratings_for_interacted_movies.melt(id_vars='movieId', value_vars=['actual_rating', 'predicted_ra
sns.displot(data=df, x='value', hue='variable', kde=True);

```



```
In [119]: # Instantiating Reader scale with expected rating scale
reader = Reader(rating_scale=(0, 5))

# Loading the rating dataset
data = Dataset.load_from_df(rating[['userId', 'movieId', 'rating']], reader)

# Splitting the data into train and test dataset
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)
```

Precision and Recall @ k

RMSE is not the only metric we can use here. We can also examine two fundamental measures, precision and recall. We also add a parameter k which is helpful in understanding problems with multiple rating outputs.

Precision@k - It is the **fraction of recommended items that are relevant in top k predictions**. Value of k is the number of recommendations to be provided to the user. One can choose a variable number of recommendations to be given to a unique user.

Recall@k - It is the **fraction of relevant items that are recommended to the user in top k predictions**.

Recall - It is the **fraction of actually relevant items that are recommended to the user** i.e. if out of 10 relevant movies, 6 are recommended to the user then recall is 0.60. Higher the value of recall better is the model. It is one of the metrics to do the performance assessment of classification models.

Precision - It is the **fraction of recommended items that are relevant actually** i.e. if out of 10 recommended items, 6 are found relevant by the user then precision is 0.60. The higher the value of precision better is the model. It is one of the metrics to do the performance assessment of classification models.

See the Precision and Recall @ k section of your notebook and follow the instructions to compute various precision/recall values at various values of k.

To know more about precision recall in Recommendation systems refer to these links :

<https://surprise.readthedocs.io/en/stable/FAQ.html>

https://medium.com/@m_n_malaeb/recall-and-precision-at-k-for-recommender-systems-618483226c54

Question6: Compute the precision and recall, for each of the 6 models, at k = 5 and 10. This is 6 x 2 = 12 numerical values? (4 marks)

```
In [120]: # Function can be found on surprise documentation FAQs
def precision_recall_at_k(predictions, k=10, threshold=3.5):
    """Return precision and recall at k metrics for each user"""

    # First map the predictions to each user.
    user_est_true = defaultdict(list)
    for uid, _, true_r, est, _ in predictions:
        user_est_true[uid].append((est, true_r))
```

```

precisions = dict()
recalls = dict()
for uid, user_ratings in user_est_true.items():

    # Sort user ratings by estimated value
    user_ratings.sort(key=lambda x: x[0], reverse=True)

    # Number of relevant items
    n_rel = sum((true_r >= threshold) for (_, true_r) in user_ratings)

    # Number of recommended items in top k
    n_rec_k = sum((est >= threshold) for (est, _) in user_ratings[:k])

    # Number of relevant and recommended items in top k
    n_rel_and_rec_k = sum(((true_r >= threshold) and (est >= threshold))
                          for (est, true_r) in user_ratings[:k])

    # Precision@K: Proportion of recommended items that are relevant
    # When n_rec_k is 0, Precision is undefined. We here set it to 0.

    precisions[uid] = n_rel_and_rec_k / n_rec_k if n_rec_k != 0 else 0

    # Recall@K: Proportion of relevant items that are recommended
    # When n_rel is 0, Recall is undefined. We here set it to 0.

    recalls[uid] = n_rel_and_rec_k / n_rel if n_rel != 0 else 0

return precisions, recalls

```

```

In [123]: from surprise import NMF

# A basic cross-validation iterator.
kf = KFold(n_splits=5)

# Make list of k values
K = [5, 10]

# Make list of models
models = [SVD(n_factors=50, n_epochs=10, biased=True, lr_all=0.01, reg_all=0.1),
          NMF(n_factors=50, n_epochs=10, biased=True, reg_pu=0.06, reg_qi=0.06, reg_bu=0.02, reg_bi=0.02)]

for k in K:
    for model in models:
        print('> k={}, model={}'.format(k, model.__class__.__name__))
        p = []
        r = []
        for trainset, testset in kf.split(data):
            model.fit(trainset)
            predictions = model.test(testset, verbose=False)
            precisions, recalls = precision_recall_at_k(predictions, k=k, threshold=3.5)

            # Precision and recall can then be averaged over all users
            p.append(sum(prec for prec in precisions.values()) / len(precisions))
            r.append(sum(rec for rec in recalls.values()) / len(recalls))

        print('-----> Precision: ', round(sum(p) / len(p), 3))
        print('-----> Recall: ', round(sum(r) / len(r), 3))

```

```

> k=5, model=SVD
-----> Precision:  0.746
-----> Recall:   0.385
> k=5, model=NMF
-----> Precision:  0.582
-----> Recall:   0.274
> k=10, model=SVD
-----> Precision:  0.729
-----> Recall:   0.521
> k=10, model=NMF
-----> Precision:  0.601
-----> Recall:   0.439

```

Question 7 (5 Marks)

7.1 Compare the results from the base line user-user and item-item based models.

7.2 How do these baseline models compare to each other with respect to the tuned user-user and item-item models?

7.3 The matrix factorization model is different from the collaborative filtering models. Briefly describe this difference. Also, compare the RMSE and precision recall for the models.

7.4 Does it improve? Can you offer any reasoning as to why that might be?

Write your Answer here: __

The user-user and item-item based models have similar performance with respect to RMSE and precision-recall metrics.

The tuned user-user and item-item models outperform the baseline models with respect to both RMSE and precision-recall metrics. The tuned item-item model has a better recall, while the tuned user-user model has a better precision.

The matrix factorization model is different from the collaborative filtering models because it learns the underlying features of the users and items to make predictions, whereas collaborative filtering models use the raw ratings data.

Yes, the matrix factorization model improves upon the baseline models. This is likely because the matrix factorization model is able to capture more complex relationships between users and items, leading to more accurate predictions.

Conclusions

In this case study, we saw three different ways of building recommendation systems:

- rank-based using averages
- similarity-based collaborative filtering
- model-based (matrix factorization) collaborative filtering

We also understood advantages/disadvantages of these recommendation systems and when to use which kind of recommendation systems. Once we build these recommendation systems, we can use **A/B Testing** to measure the effectiveness of these systems.

Here is an article explaining how [Amazon use A/B Testing](#) to measure effectiveness of its recommendation systems.