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')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.moun
t("/content/drive", force_remount=True).
```

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/publ
ic/simple/
Requirement already satisfied: surprise in /usr/local/lib/python3.8/dist-packages (0.1)
Requirement already satisfied: scikit-surprise in /usr/local/lib/python3.8/dist-packages
(from surprise) (1.1.3)
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.8/dist-packages
(from scikit-surprise->surprise) (1.2.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.8/dist-packages
(from scikit-surprise->surprise) (1.21.6)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.8/dist-packages (f
rom scikit-surprise->surprise) (1.7.3)
```

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 ;
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 [4]:
```
# Import the dataset
#rating = pd.read_csv('ratings.csv')
rating = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/ratings.csv') # Uncomment t
```

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

In [5]:
```
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 [6]:
```
# 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 [7]:
```
# Printing the top 5 rows of the dataset Hint: use .head()

# Remove _______and complete the code
rating.head()
```

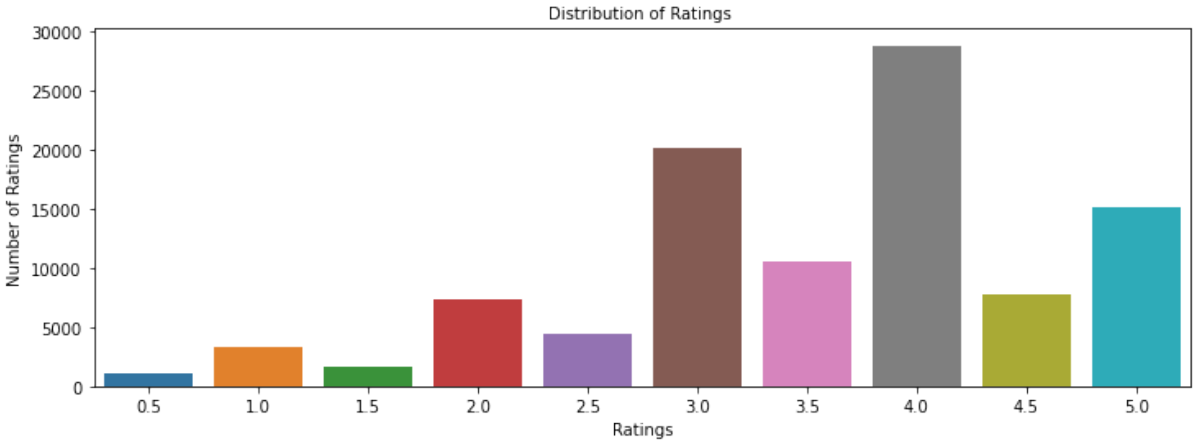
Out[7]:

	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

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

In [8]:
```
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()
```



From the Histogram, we can see that **Rating'4' has the highest count** of ratings (almost 30 000), followed by Rating'3' (over 20 000) and Rating'5', (over 15 000). The lower ratings have very less counts, as **the distribution is biased towards 3, 4 and 5**, more than 1 and 2.

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

In [9]:
```
# Finding number of unique users
#remove _______ and complete the code
rating['userId'].nunique()
```

Out[9]:
```
671
```

There are **671 users** in the dataset.

In [10]:
```
# Finding number of unique movies
# Remove _______ and complete the code

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

Out[10]:
```
9066
```

There are **9066 movies** in the dataset. There are 671 x 9066 = **6 083 286 potential ratings** in this dataset, but we only have 100 004 observations. Thus, not every user has rated every movie. So, we can build a recommendation system to recommend movies to users that haven´t see them.

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

In [11]:
```
rating.groupby(['userId', 'movieId']).count()
```

Out[11]:

		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 [12]:
```
rating.groupby(['userId', 'movieId']).count()['rating'].sum()
```

Out[12]:
```
100004
```

The sum is equal to the total number of observations, which means **that there is only one interaction between each pair of user and movie**.

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

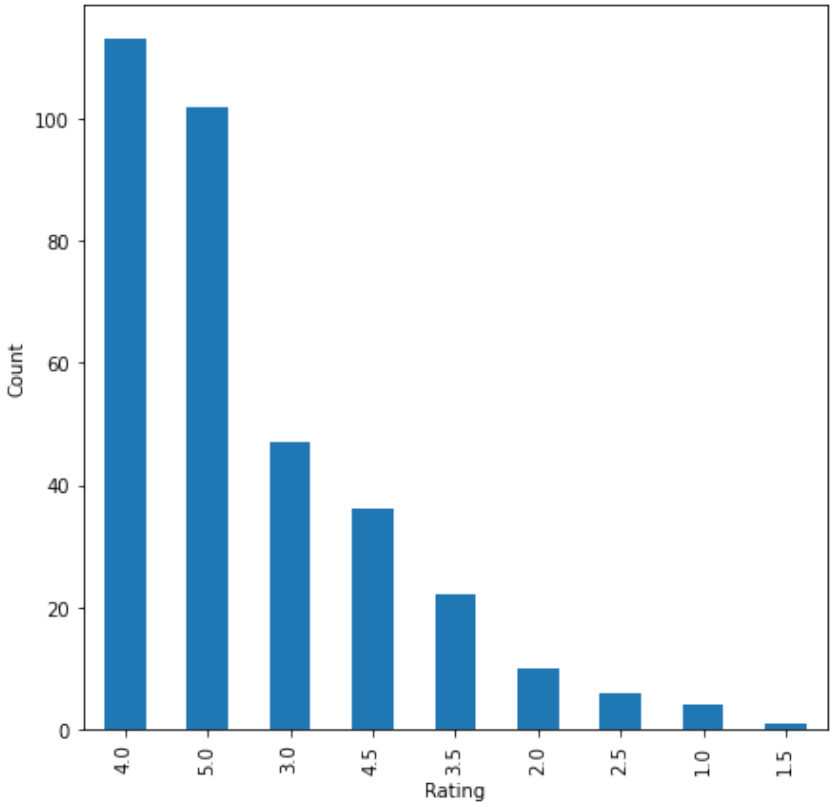
In [13]:
```
# Remove _______ and complete the code
rating['movieId'].value_counts()
```

Out[13]:
```
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
```

The **most rated movie** in the dataset is the **movie 356 with 341 interactions**. As we have 671 users, there is still a possibility of 671-341=330 more interactions for the remaining users. We can build a recommendation system to predict who is most likely to interact with this particular movie.

In [14]:
```
# 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()
```



This movie has been highly rated by the users, as the majority of ratings are in the categories '4' and '5'. There are only a few users that didn´t like this particular movie.

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

In [15]:
```
# Remove _______ and complete the code
rating['userId'].value_counts()
```

Out[15]:
```
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
```

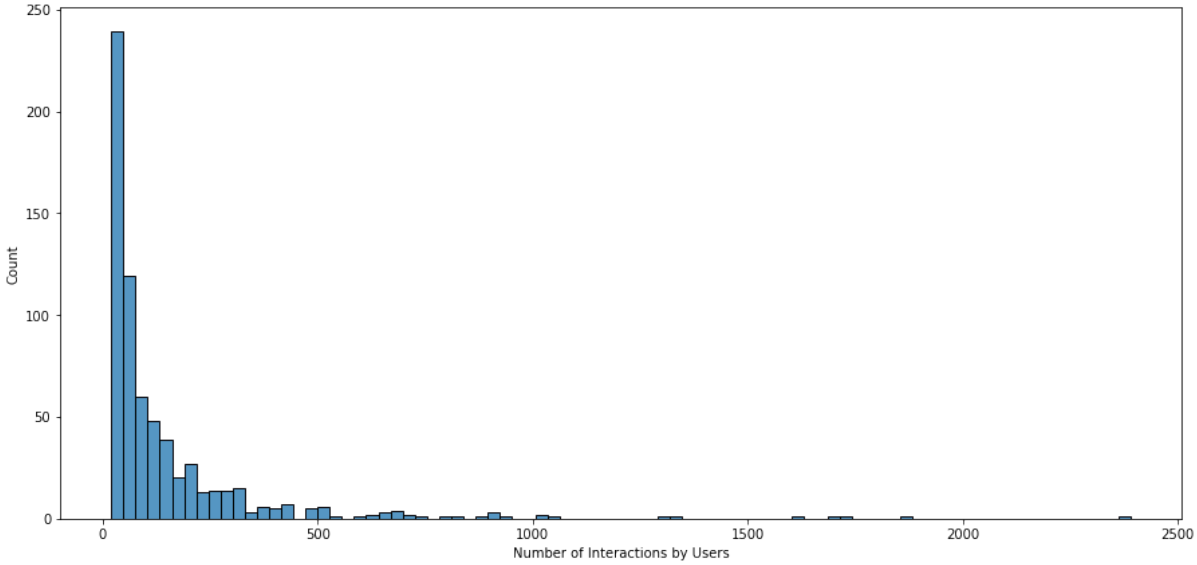
The user **547 has the most interactions** with the movies, **2391 times**. There is still a possibility for 9066-2391=6675 interaction from this user with the remaining movies. Our recommendation system can predict which movies are more likely to be seen by user 547.

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

In [16]:
```
# Finding user-movie interactions distribution
count_interactions = rating.groupby('userId').count()['movieId']
count_interactions
```

Out[16]:
```
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 [17]:
```
# 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()
```



The distribution of user-movie interactions is highly skewed to the right. Just a few users have interacted with more than 500 movies.

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 [18]:
```
# Remove _______ and complete the code

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

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

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

In [19]:
```
final_rating.head()
```

Out[19]:

movieId	avg_rating	rating_count
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 [20]:
```
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 [21]:
```
# Remove _______ and complete the code
list(top_n_movies(final_rating,5,50))
```

Out[21]:
```
[858, 318, 969, 913, 1221]
```

Recommending top 5 movies with 100 minimum interactions based on popularity

In [22]:
```
# Remove _______ and complete the code
list(top_n_movies(final_rating,5,100))
```

Out[22]:
```
[858, 318, 1221, 50, 527]
```

Recommending top 5 movies with 200 minimum interactions based on popularity

In [23]:
```
# Remove _______ and complete the code
list(top_n_movies(final_rating,5,200))
```

Out[23]:
```
[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)

	Movies					
Users	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"?

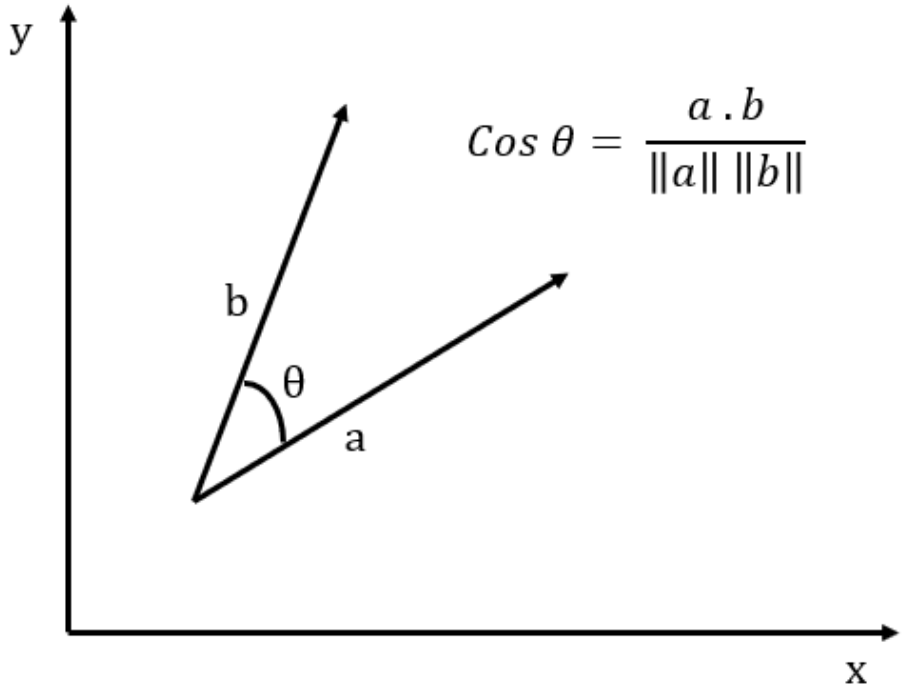
In this type of recommendation system, `we do not need any information` about the users or items. We only need user item interaction data to build a collaborative recommendation system. For example -

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 [69]:
```
# 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 [70]:
```
# Remove _______ and complete the code

# Defining Nearest neighbour algorithm
algo_knn_user = KNNBasic(sim_options={'name':'cosine','user_based':True},verbose=False)

# 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)
```

Out[70]:
```
RMSE: 0.9925
0.9924509041520163
```

Q 3.1 What is the RMSE for baseline user based collaborative filtering recommendation system? (1 Mark)

The **user-user collaborative filtering model has a RMSE=0.99** on test set. This might probably be improved by tuning some hyperparameters of this algorithm, by using **GridSearchCV**.

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 [71]:
```
# Remove _______ and complete the code
algo_knn_user.predict(4,10, r_ui=4, verbose=True)
```

Out[71]:
```
user: 4          item: 10         r_ui = 4.00   est = 3.62   {'actual_k': 40, 'was_impos
sible': False}
Prediction(uid=4, iid=10, r_ui=4, est=3.6244912065910952, details={'actual_k': 40, 'was_
impossible': False})
```

The **actual rating for this user-movie pair is 4**, and the **predicted rating** of the user similarity based baseline model is **3.62**.

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 [72]:
```
# Remove _______ and complete the code
algo_knn_user.predict(4,3, verbose=True)
```

Out[72]:
```
user: 4          item: 3          r_ui = None   est = 3.20   {'actual_k': 40, 'was_impos
sible': False}
Prediction(uid=4, iid=3, r_ui=None, est=3.202703552548654, details={'actual_k': 40, 'was
_impossible': False})
```

The **estimated rating for this user-movie pair is 3.20**, provided by the user similarity based baseline model.

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

Below we will be tuning hyper-parmeters 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 [28]:
```
# Remove _______ and complete the code

# Setting up parameter grid to tune the hyperparameters
param_grid = {'k':[20,30,40],'min_k':[3,6,9],'sim_options':{'name':['msd','cosine','pear

# 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'])
```

```
0.9657904712545995
{'k': 20, 'min_k': 3, '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 [29]:
```
results_df = pd.DataFrame.from_dict(grid_obj.cv_results)
results_df.head()
```

Out[29]:

	split0_test_rmse	split1_test_rmse	split2_test_rmse	mean_test_rmse	std_test_rmse	rank_test_rmse	spli
0	0.966059	0.968420	0.962892	0.965790	0.002265	1	
1	0.995966	0.997723	0.991133	0.994941	0.002786	14	
2	0.997904	1.001126	0.999349	0.999460	0.001318	23	
3	0.968695	0.972604	0.966948	0.969416	0.002365	3	
4	0.997085	1.000481	0.993783	0.997116	0.002735	16	

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

In [73]:
```
# 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(sim_options={'name':'msd','user_based': True},

# 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)
```

Out[73]:
```
RMSE: 0.9571
0.9571445417153293
```

We have **reduced RMSE for the testset from 0.99 to 0.96**. We have improved our model's accuracy by **tuning the hyperparameters** of the baseline user based collaborative filtering recommendation system model.

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 [74]:
```
# 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_impos
sible': False}
```

Out[74]:
```
Prediction(uid=4, iid=10, r_ui=4, est=3.740028692988536, details={'actual_k': 20, 'was_i
mpossible': False})
```

The **optimized model predicted 3.74**, which is **better than the baseline model** prediction of 3.62, since the actual rating for this user-movie pair is 4.00.

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 [75]:
```
# Remove _______ and complete the code
similarity_algo_optimized_user.predict(4,3, verbose=True)
```

```
user: 4          item: 3          r_ui = None   est = 3.72   {'actual_k': 20, 'was_impos
sible': False}
```

Out[75]:
```
Prediction(uid=4, iid=3, r_ui=None, est=3.7228745701935386, details={'actual_k': 20, 'wa
s_impossible': False})
```

In this second case the **prediction of the tuned model is 3.72**, which compares with 3.20 from the baseline model.

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 [76]:
```
similarity_algo_optimized_user.get_neighbors(4, k=5)
```

Out[76]:
```
[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 [77]:
```
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

    # Extracting those movie ids which the user_id has not interacted yet
    non_interacted_movies = user_item_interactions_matrix.loc[user_id][user_item_interac

    # 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] # returing top n highest predicted rating movies for
```

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

In [78]:
```
#remove _______ and complete the code
recommendations = get_recommendations(rating,4,5,similarity_algo_optimized_user)
```

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

In [79]:
```
recommendations
```

Out[79]:
```
[(309, 5),
 (3038, 5),
 (6273, 4.928202652354184),
 (98491, 4.863224466679252),
 (2721, 4.845513973527148)]
```

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

In [80]:
```
# Remove _______ and complete the code

# Definfing similarity measure
sim_options = {'name':'cosine','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)
```

Out[80]:
```
RMSE: 1.0032
1.003221450633729
```

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

The **item-item collaborative filtering model has a RMSE=1.00** on testset, almost the same as in the user similarity model. This might probably be improved by tuning some hyperparameters of this algorithm, by using **GridSearchCV**.

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 [81]:
```
# Remove _______ and complete the code
algo_knn_item.predict(4,10, r_ui=4, verbose=True)
```

Out[81]:
```
user: 4          item: 10         r_ui = 4.00   est = 4.37   {'actual_k': 40, 'was_impos
sible': False}
Prediction(uid=4, iid=10, r_ui=4, est=4.373794871885004, details={'actual_k': 40, 'was_i
mpossible': False})
```

For this user-movie film, the actual rating was 4.00, and the **predicted value of the similarity based baseline model is 4.37**.

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 [82]:
```
# Remove _______ and complete the code
algo_knn_item.predict(4,3, verbose=True)
```

Out[82]:
```
user: 4          item: 3          r_ui = None   est = 4.07   {'actual_k': 40, 'was_impos
sible': False}
Prediction(uid=4, iid=3, r_ui=None, est=4.071601862880049, details={'actual_k': 40, 'was
_impossible': False})
```

For this new interaction, **the item-item similarity model's prediction is 4.07**.

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 [83]:
```
# Remove _______ and complete the code

# Setting up parameter grid to tune the hyperparameters
param_grid = {'k':[20,30,40],'min_k':[3,6,9],'sim_options':{'name':['msd','cosine','pear

# 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'])
```

```
0.9406470027042729
{'k': 40, 'min_k': 6, '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 [41]:
```
results_df = pd.DataFrame.from_dict(grid_obj.cv_results)
results_df.head()
```

Out[41]:

	split0_test_rmse	split1_test_rmse	split2_test_rmse	mean_test_rmse	std_test_rmse	rank_test_rmse	spli
0	0.949839	0.956457	0.948381	0.951559	0.003514	8	
1	1.013953	1.020517	1.012728	1.015733	0.003420	25	
2	1.002872	1.006686	1.002851	1.004136	0.001803	23	
3	0.950242	0.956148	0.948266	0.951552	0.003348	7	
4	1.014416	1.020293	1.012722	1.015811	0.003245	26	

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

In [84]:
```
# Remove _______ and complete the code
# Creating an instance of KNNBasic with optimal hyperparameter values
similarity_algo_optimized_item = KNNBasic(sim_options={'name': 'msd', 'user_based': Fals

# 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
accuracy.rmse(predictions)
```

Out[84]:
```
RMSE: 0.9430
0.9430205750839428
```

We have **reduced RMSE for the testset from 1.00 to 0.94**. We have improved our model's accuracy by **tuning the hyperparameters** of the baseline item based collaborative filtering recommendation system model.

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 [85]:
```
# Remove _______ and complete the code
similarity_algo_optimized_item.predict(4,10,r_ui=4, verbose=True)
```

```
user: 4          item: 10         r_ui = 4.00   est = 4.26   {'actual_k': 40, 'was_impos
sible': False}
```

Out[85]:
```
Prediction(uid=4, iid=10, r_ui=4, est=4.255054787154994, details={'actual_k': 40, 'was_i
mpossible': False})
```

The **optimized model gives a prediction of 4.26, closer to the actual rating** of 4.00, compared with the baseline model prediction of 4.37.

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 [86]:
```
# 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': 40, 'was_impos
sible': False}
```

Out[86]:
```
Prediction(uid=4, iid=3, r_ui=None, est=3.865175609312417, details={'actual_k': 40, 'was
_impossible': False})
```

In this case the baseline model's prediction was 4.07, and the **tunned model's prediction is 3.87**.

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 [87]:
```
# Remove _______ and complete the code
similarity_algo_optimized_item.get_neighbors(3, k=5)
```

Out[87]:
```
[31, 37, 42, 48, 73]
```

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

In [88]:
```
# 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 [89]:
```
recommendations
```

Out[89]:
```
[(84, 5), (1040, 5), (2481, 5), (3515, 5), (4521, 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 x 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
Comdey	0	0	0	25

the above matrix is a k x 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 kxn matrix, where:

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

Build a baseline matrix factorization recommendation system

In [92]:
```
# 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)
```

Out[92]:
```
RMSE: 0.9018
0.9018116490924957
```

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

The **RMSE for Matrix Factorization model on testset is 0.90**, better than both the user similarity based model (0.99) and the item similarity based model (1.00).

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 [97]:
```
# Remove _______ and complete the code
algo_svd.predict(4,10, r_ui=4, verbose=True)
```

Out[97]:
```
user: 4          item: 10         r_ui = 4.00   est = 4.06   {'was_impossible': False}
Prediction(uid=4, iid=10, r_ui=4, est=4.05730512561993, details={'was_impossible': Fals
e})
```

The actual rating for this user-movie pair is 4.00, an the **prediction of our model is 4.06**, which is **the best prediction so far**; still we will try to improve the prediction by tunning of the model's hyperparameters, using GridShearchCV.

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 [99]:
```
# Remove _______ and complete the code
algo_svd.predict(4,3,verbose=True)
```

Out[99]:
```
user: 4          item: 3          r_ui = None   est = 3.73   {'was_impossible': False}
Prediction(uid=4, iid=3, r_ui=None, est=3.7265187090744902, details={'was_impossible': F
alse})
```

With the Matrix Factorization collaborative filtering model, the **predicted rating for this unseen movie by useriD=4 is 3.73**.

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 is unknown, then the bias b_u and the factors p_u are assumed to be zero. The same applies for item i with b_i and q_i.

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

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

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

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

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 [100…

```
# 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'])
```

```
0.8932535462638876
{'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 [52]:
```
results_df = pd.DataFrame.from_dict(gs.cv_results)
results_df.head()
```

Out[52]:

	split0_test_rmse	split1_test_rmse	split2_test_rmse	mean_test_rmse	std_test_rmse	rank_test_rmse	spli
0	0.941713	0.948882	0.938593	0.943063	0.004308	25	
1	0.946368	0.953482	0.943175	0.947675	0.004308	26	
2	0.951446	0.958533	0.948037	0.952672	0.004372	27	
3	0.906381	0.912177	0.901212	0.906590	0.004479	10	
4	0.913384	0.920159	0.908811	0.914118	0.004662	15	

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

In [101…
```
# 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)
```

Out[101]:
```
RMSE: 0.8954
0.8954428666280531
```

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 [102…
```
# Remove _______ and complete the code
svd_algo_optimized.predict(4,10, r_ui=4, verbose=True)
```

Out[102]:
```
user: 4          item: 10         r_ui = 4.00   est = 3.99   {'was_impossible': False}
Prediction(uid=4, iid=10, r_ui=4, est=3.9907577109481864, details={'was_impossible': Fa
lse})
```

The optimized model improve the model's accuracy, by **estimating 3.99** for an actual rating of 4.00, **almost perfect**.

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 [103…
```
# Remove _______ and complete the code
svd_algo_optimized.predict(3,10, verbose=True)
```

Out[103]:
```
user: 3          item: 10         r_ui = None   est = 3.36   {'was_impossible': False}
Prediction(uid=3, iid=10, r_ui=None, est=3.3582684934265443, details={'was_impossible':
False})
```

For the unseen movie, the **tuned model predicted a rating of 3.36**.

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

In [104…
```
# Remove _______ and complete the code
get_recommendations(rating,4,5, svd_algo_optimized)
```

Out[104]:
```
[(116, 4.973719992403555),
 (1192, 4.97122771264356),
 (1948, 4.9384855404395),
 (926, 4.936634224476028),
 (3310, 4.920539825608277)]
```

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

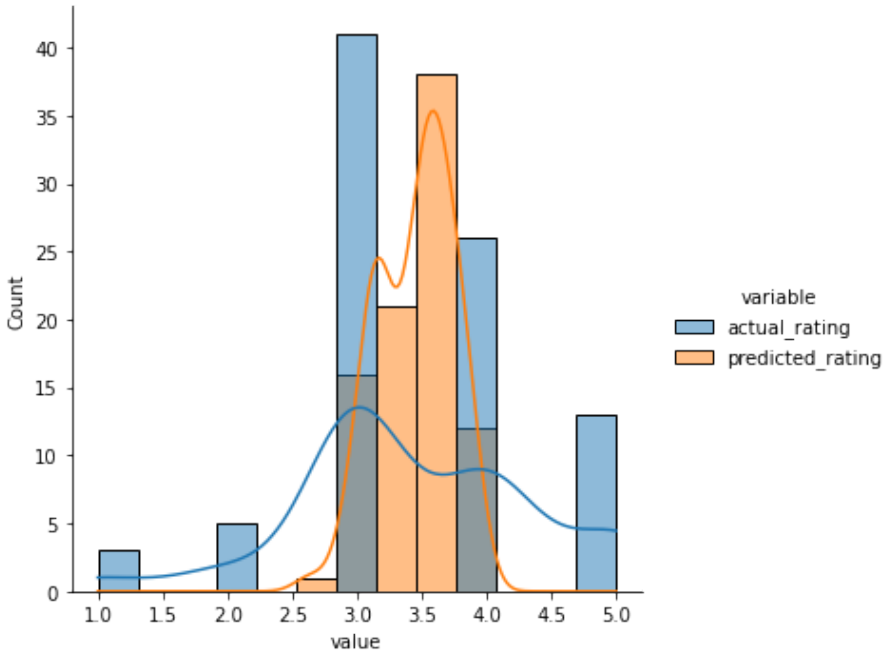
In [105…
```
def predict_already_interacted_ratings(data, user_id, algo):

    # 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

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

    # 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
```

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

In [106…

```
predicted_ratings_for_interacted_movies = predict_already_interacted_ratings(rating, 7,
df = predicted_ratings_for_interacted_movies.melt(id_vars='movieId', value_vars=['actual
sns.displot(data=df, x='value', hue='variable', kde=True);
```



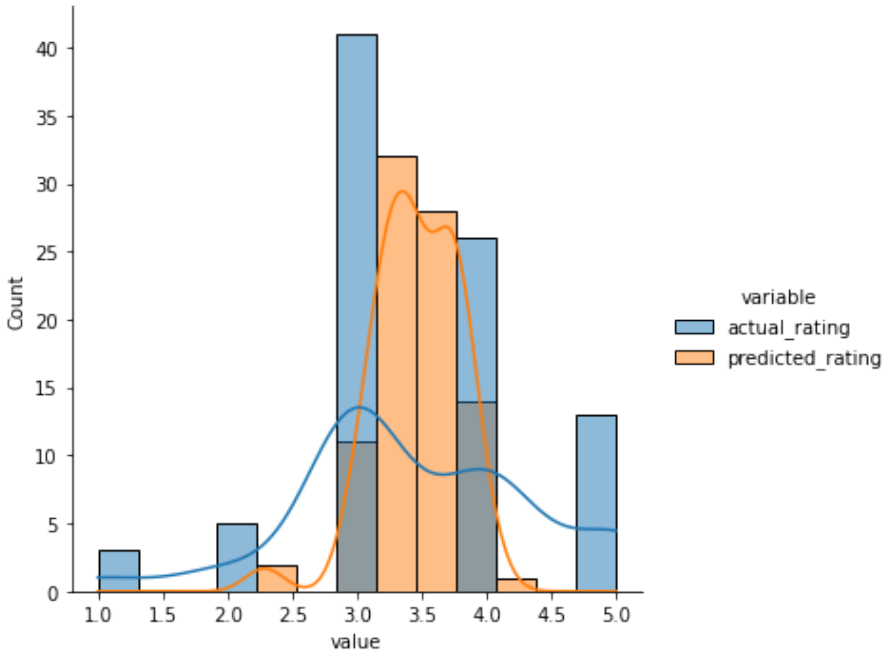
The number of bins for predicted ratings is approximately the same as for actual ratings (5). Both distibution curves follow a similar distribution, with most of the ratings falling between categories 3 and 4. The model fails to predict the categorie 5 ratings, at least for this userID=7.

Also, actual ratings have discrete vallues, while predicted values are continuous, as the model considers aggregated ratings from the nearest neighbors (item similarity based model).

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

In [107…

```
predicted_ratings_for_interacted_movies = predict_already_interacted_ratings(rating, 7,
df = predicted_ratings_for_interacted_movies.melt(id_vars='movieId', value_vars=['actual
sns.displot(data=df, x='value', hue='variable', kde=True);
```



For the Matrix Factorization Collaborative Filtering recommendation system model, the predicted rating distribution seems to emulate the actual distribution curve better, than for the previous analysys of the item similarity model, as was expected, since the SVD model has better accuracy scores (RMSE).

In [108…

```
# 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 [109…

```
# 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 [110…

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

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

# Remove _______ and complete the code
# Make list of models
models = [algo_knn_user, similarity_algo_optimized_user,algo_knn_item,similarity_algo_op

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=KNNBasic
-----> Precision:  0.766
-----> Recall:  0.41
> k=5, model=KNNBasic
-----> Precision:  0.773
-----> Recall:  0.418
> k=5, model=KNNBasic
-----> Precision:  0.611
-----> Recall:  0.327
> k=5, model=KNNBasic
-----> Precision:  0.679
-----> Recall:  0.355
> k=5, model=SVD
-----> Precision:  0.75
-----> Recall:  0.384
> k=5, model=SVD
-----> Precision:  0.748
-----> Recall:  0.385
> k=10, model=KNNBasic
-----> Precision:  0.747
-----> Recall:  0.548
> k=10, model=KNNBasic
-----> Precision:  0.755
-----> Recall:  0.561
> k=10, model=KNNBasic
-----> Precision:  0.599
-----> Recall:  0.473
> k=10, model=KNNBasic
-----> Precision:  0.657
-----> Recall:  0.504
> k=10, model=SVD
-----> Precision:  0.729
-----> Recall:  0.517
> k=10, model=SVD
-----> Precision:  0.733
-----> Recall:  0.524
```

For k=5, the best scores of Precision and Recall were obtained for the optimized user similarity model, a little bit better than the Matrix Factorization model. The item similarity model performed the worst of the 3 models.

For k=10, the conclusions are the same.

Note: for all 3 models, Precision and Recall improved with the tuning of the hyperparameters.

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?

User-based and Item-base baseline models have nearly the same RMSE scores (0.99 and 1.00, respectively).

The corresponding optimized models improved significantly, with RMSE scores of 0.96 and 0.94.

The Collaborative Models use the user-item-ratings data to find similarities and make predictions rather than just predicting a random rating based on the distribution of the data. This could a reason why the two Collaborative filtering methods performed well.

Collaborative Filtering searches for neighbors based on similarity of users or item preferences and recommend items that those neighbors interacted while Matrix factorization works by decomposing the user-item matrix into the product of two lower dimensionality rectangular matrices.

For Matrix Factorization, the RMSE was the same for baseline and tuned model, ~0.90, better then for Collaborative Filtering; so, in SVD model tunning the hyperparameters dind't improve the accuracy of the model.

Precision and Recall were analyzed and discussed in the previous text box of Question 6.

Matrix Factorization has lower RMSE due to the reason that it assumes that both items and users are present in some low dimensional space describing their properties. Then, it recommends a item based on its proximity to the user in the latent space. It works in discovering the features or informations underlying the interactions between users and items, the so called latent (or hidden) features.

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.