Project: Amazon Product Recommendation System

Marks: 40

Welcome to the project on Recommendation Systems. We will work with the Amazon product reviews dataset for this project. The dataset contains ratings of different electronic products. It does not include information about the products or reviews to avoid bias while building the model.

Context:

Today, information is growing exponentially with volume, velocity and variety throughout the globe. This has lead to information overload, and too many choices for the consumer of any business. It represents a real dilemma for these consumers and they often turn to denial. Recommender Systems are one of the best tools that help recommending products to consumers while they are browsing online. Providing personalized recommendations which is most relevant for the user is what's most likely to keep them engaged and help business.

E-commerce websites like Amazon, Walmart, Target and Etsy use different recommendation models to provide personalized suggestions to different users. These companies spend millions of dollars to come up with algorithmic techniques that can provide personalized recommendations to their users.

Amazon, for example, is well-known for its accurate selection of recommendations in its online site. Amazon's recommendation system is capable of intelligently analyzing and predicting customers' shopping preferences in order to offer them a list of recommended products. Amazon's recommendation algorithm is therefore a key element in using AI to improve the personalization of its website. For example, one of the baseline recommendation models that Amazon uses is item-to-item collaborative filtering, which scales to massive data sets and produces high-quality recommendations in real-time.

Objective:

You are a Data Science Manager at Amazon, and have been given the task of building a recommendation system to recommend products to customers based on their previous ratings for other products. You have a collection of labeled data of Amazon reviews of products. The goal is to extract meaningful insights from the data and build a recommendation system that helps in recommending products to online consumers.

Dataset:

The Amazon dataset contains the following attributes:

- **userId:** Every user identified with a unique id
- **productId:** Every product identified with a unique id
- **Rating:** The rating of the corresponding product by the corresponding user
- **timestamp:** Time of the rating. We **will not use this column** to solve the current problem

Note: The code has some user defined functions that will be usefull while making recommendations and measure model performance, you can use these functions or can create your own functions.

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 project.

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

Installing surprise library

In [106…
```
#!pip install surprise
!pip install scikit-surprise
```

```
Defaulting to user installation because normal site-packages is not writeable
Looking in links: /usr/share/pip-wheels
Requirement already satisfied: surprise in /home/cca15a05-6a3a-4368-9fd9-87213cd22414/.local/lib/python3.11/site
-packages (0.1)
Requirement already satisfied: scikit-surprise in /home/cca15a05-6a3a-4368-9fd9-87213cd22414/.local/lib/python3.
11/site-packages (from surprise) (1.1.4)
Requirement already satisfied: joblib>=1.2.0 in /opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site
-packages (from scikit-surprise->surprise) (1.2.0)
Requirement already satisfied: numpy>=1.19.5 in /opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site
-packages (from scikit-surprise->surprise) (1.24.3)
Requirement already satisfied: scipy>=1.6.0 in /opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-
packages (from scikit-surprise->surprise) (1.11.1)
```

Importing the necessary libraries and overview of the dataset

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

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

import matplotlib.pyplot as plt                # Basic library for data visualization
import seaborn as sns                          # Slightly advanced library for data visualization

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

from sklearn.metrics import mean_squared_error  # A performance metrics in sklearn
```

Loading the data

- Import the Dataset
- Add column names ['user_id', 'prod_id', 'rating', 'timestamp']
- Drop the column timestamp
- Copy the data to another DataFrame called **df**

In [4]:
```
# Import the dataset
df = pd.read_csv('ratings_Electronics.csv', header = None) # There are no headers in the data file
df.columns = ['user_id', 'prod_id', 'rating', 'timestamp'] # Adding column names
df = df.drop('timestamp', axis = 1) # Dropping timestamp
df_copy = df.copy(deep = True) # Copying the data to another DataFrame
```

As this dataset is very large and has 7,824,482 observations, it is not computationally possible to build a model using this. Moreover, many users have only rated a few products and also some products are rated by very few users. Hence, we can reduce the dataset by considering certain logical assumptions.

Here, we will be taking users who have given at least 50 ratings, and the products that have at least 5 ratings, as when we shop online we prefer to have some number of ratings of a product.

In [5]:
```
# Get the column containing the users
users = df.user_id

# Create a dictionary from users to their number of ratings
ratings_count = dict()

for user in users:
    # If we already have the user, just add 1 to their rating count
    if user in ratings_count:
        ratings_count[user] += 1
    # Otherwise, set their rating count to 1
    else:
        ratings_count[user] = 1

#ratings_count = df['user_id'].value_counts().to_dict()
```

In [6]:
```
# We want our users to have at least 50 ratings to be considered
RATINGS_CUTOFF = 50

remove_users = []
for user, num_ratings in ratings_count.items():
    if num_ratings < RATINGS_CUTOFF:
        remove_users.append(user)

df = df.loc[ ~ df.user_id.isin(remove_users)]
```

In [7]:
```
# Get the column containing the products
prods = df.prod_id

# Create a dictionary from products to their number of ratings
```

```
ratings_count = dict()

for prod in prods:

    # If we already have the product, just add 1 to its rating count
    if prod in ratings_count:
        ratings_count[prod] += 1

    # Otherwise, set their rating count to 1
    else:
        ratings_count[prod] = 1
```

In [8]:
```
# We want our item to have at least 5 ratings to be considered
RATINGS_CUTOFF = 5

remove_users = []

for user, num_ratings in ratings_count.items():
    if num_ratings < RATINGS_CUTOFF:
        remove_users.append(user)

df_final = df.loc[~ df.prod_id.isin(remove_users)]
```

In [9]:
```
# Print a few rows of the imported dataset
df_final.head()
```

Out[9]:

	user_id	prod_id	rating
1310	A3LDPF5FMB782Z	1400501466	5.0
1322	A1A5KUIIIHFF4U	1400501466	1.0
1335	A2XIOXRRYX0KZY	1400501466	3.0
1451	AW3LX47IHPFRL	1400501466	5.0
1456	A1E3OB6QMBKRYZ	1400501466	1.0

Exploratory Data Analysis

Shape of the data

Check the number of rows and columns and provide observations.

In [10]:
```
# Check the number of rows and columns and provide observations
df_final.shape
```

Out[10]:
```
(65290, 3)
```

Observations:

- The dataset has 65290 rows and 3 columns.

Data types

In [11]:
```
# Check Data types and provide observations
df_final.dtypes
```

Out[11]:
```
user_id     object
prod_id     object
rating     float64
dtype: object
```

Observations:

- User ID and Product ID is object data type, and rating is numeric type.

Checking for missing values

In [12]:
```
# Check for missing values present and provide observations
print(df_final.isnull().sum())
```

```
user_id    0
prod_id    0
rating     0
dtype: int64
```

Observations:

- No missing value exist.

Summary Statistics

In [13]:
```
# Summary statistics of 'rating' variable and provide observations
df_final.describe().T
```

Out[13]:

	count	mean	std	min	25%	50%	75%	max
rating	65290.0	4.294808	0.988915	1.0	4.0	5.0	5.0	5.0

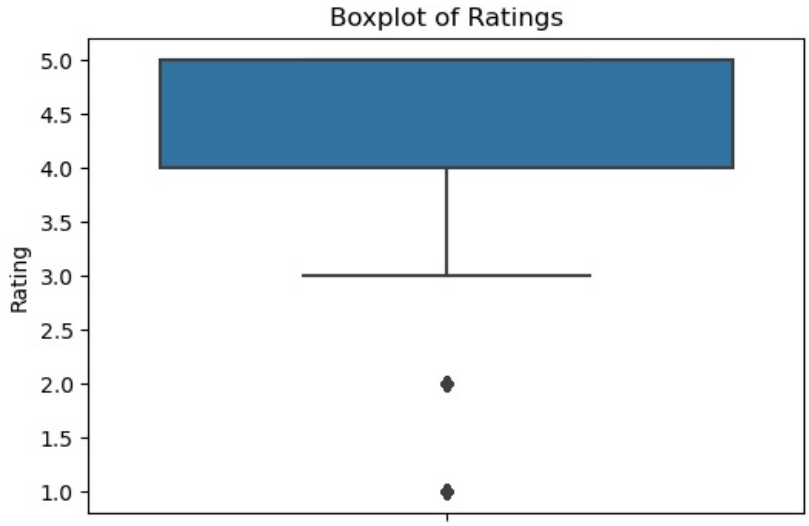
Observations:

- The average ratings is approximately 4.295. With max rating as5, it means users generally have positive experience in most Amazon products.
- The standard derivation is about 0.989, indicating that ratings concentrate around the mean rating 4.295.
- While the average mean is high compared with max rating, the ratings has a large range of 1 to 5, suggesting product ratings could vary in large range.

Checking the rating distribution

In [22]:
```
# Create a boxplot for the 'rating' column
plt.figure(figsize=(6, 4))  # Set the figure size
sns.boxplot(y='rating', data=df_final)

# Add title and labels
plt.title('Boxplot of Ratings')
plt.ylabel('Rating')
plt.show()
```



Observations:

- The median rating is around 4.5, indicating that most users tend to rate products positively.
- The interquartile range is within a narrow range of 4-5, indicating that user ratings concentrates closely around the median.
- The whiskers extend down to 3 and up to 5, indicating user ratings can vary significantly.
- There are outliers of 1 and 2, indicating some products are not well received.
- The distribution appears slightly right-skewed, as indicated by the longer whisker on the lower side.

Checking the number of unique users and items in the dataset

In [15]:
```
# Number of total rows in the data and number of unique user id and product id in the data
print('Total rows count: ', len(df_final))
print('Total no. of unique User ID: ', df_final["user_id"].nunique())
```

```
Total rows count:  65290
Total no. of unique User ID:  1540
```

Observations:

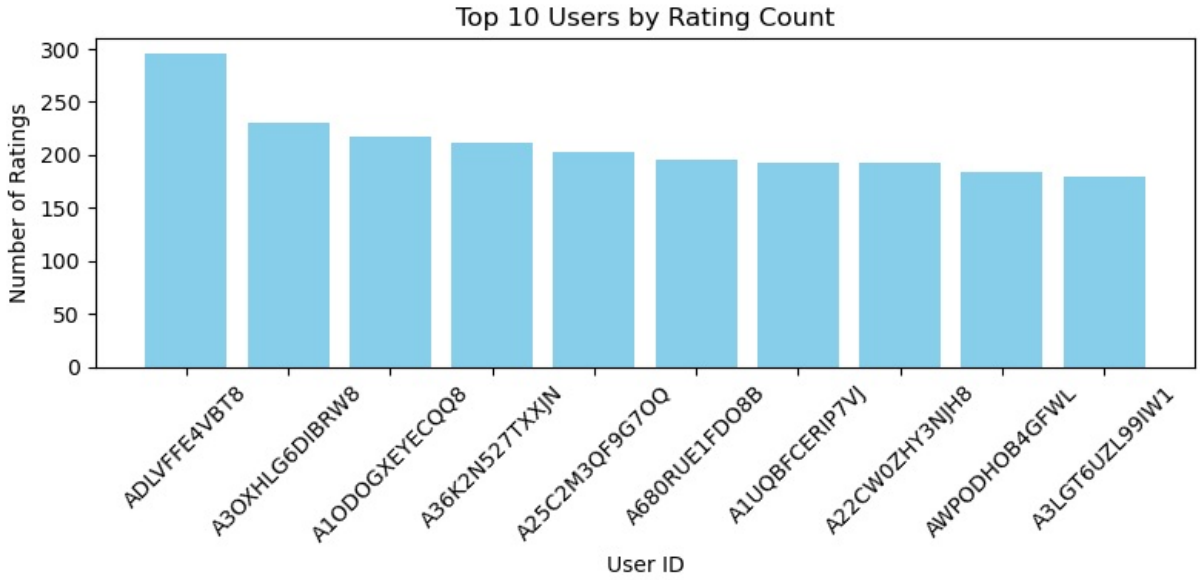
- Total row count is much higher than unqiue user id, suggesting that a user generally give multiple product ratings.

Users with the most number of ratings

In [17]:
```
# Top 10 users based on the number of ratings
top_users = df_final.groupby('user_id').size().reset_index(name='rating_count')
top_users = top_users.sort_values(by='rating_count', ascending=False)
top_10_users = top_users.head(10)
print(top_10_users)

# Plotting
plt.figure(figsize=(8, 4))
plt.bar(top_10_users['user_id'], top_10_users['rating_count'], color='skyblue')
plt.title('Top 10 Users by Rating Count')
plt.xlabel('User ID')
plt.ylabel('Number of Ratings')
plt.xticks(rotation=45)
plt.tight_layout()  # Adjust layout for better fit
plt.show()
```

```
            user_id  rating_count
1287    ADLVFFE4VBT8           295
1086  A3OXHLG6DIBRW8           230
264   A1ODOGXEYECQQ8           217
903   A36K2N527TXXJN           212
462   A25C2M3QF9G7OQ           203
1209   A680RUE1FDO8B           196
333   A1UQBFCERIP7VJ           193
431   A22CW0ZHY3NJH8           193
1508   AWPODHOB4GFWL           184
1051  A3LGT6UZL99IW1           179
```



Observations:

- The top 10 users exhibit a significant range in rating counts, varying from 179 to 295. This indicates a relatively high level of engagement among these users.
- The count plot shows that all the top-10 users actively give ratings to Amazon products. They form a major group users in rating engagement.
- The top 2 users have rating counts significantly outnumbers the others, suggesting they are either very engaged with the platform or consistently engage in product ratings.

There models of Recommendation System will be built:

- Rank Based Model
- Collaborative Filtering Model
  - User-user collaborate filtering model
  - Item-item collaborate filtering model
- Singular Value Decomposition Model

Model 1: Rank Based Recommendation System

In [66]:
```
# Calculate the average rating for each product
average_rating = df_final.groupby('prod_id')['rating'].mean().reset_index(name='average_rating')

# Calculate the count of ratings for each product
rating_count = df_final.groupby('prod_id')['rating'].count().reset_index(name='rating_count')

# Create a dataframe with calculated average and count of ratings
final_rating = pd.merge(average_rating, rating_count, on='prod_id')
```

```
# Sort the DataFrame by average rating in descending order
final_rating = final_rating.sort_values(by='average_rating', ascending=False)

# See the first five records of the "final_rating" dataset
print(final_rating.head())
```

```
        prod_id  average_rating  rating_count
5688  B00LGQ6HL8             5.0             5
2302  B003DZJQQI             5.0            14
3443  B005FDXF2C             5.0             7
5554  B00I6CVPVC             5.0             7
4810  B00B9KOCYA             5.0             8
```

In [72]:
```
# Defining a function to get the top n products based on the highest average rating and minimum interactions
def get_top_n_products(df, top_n, min_rating_count):
    # Calculate average rating and rating count for each product
    average_rating = df.groupby('prod_id')['rating'].mean().reset_index(name='average_rating')
    rating_count = df.groupby('prod_id')['rating'].count().reset_index(name='rating_count')

    # Merge the two DataFrames
    final_rating = pd.merge(average_rating, rating_count, on='prod_id')

    # Finding products with minimum number of interactions
    filtered_products = final_rating[final_rating['rating_count'] >= min_rating_count]

    # Sorting values with respect to average rating and get top n items
    return filtered_products.sort_values(by='average_rating', ascending=False).head(top_n)
```

Recommending top 5 products with 50 minimum interactions based on popularity

In [73]:
```
top_n = 5
min_interactions = 50
top_products = get_top_n_products(df_final, top_n, min_interactions)
print(f"Top {top_n} Products with {min_interactions} minimum interactions based on popularity:\n{top_products}"
```

```
Top 5 Products with 50 minimum interactions based on popularity:
         prod_id  average_rating  rating_count
1594  B001TH7GUU        4.871795            78
2316  B003ES5ZUU        4.864130           184
1227  B0019EHU8G        4.855556            90
3877  B006W8U2MU        4.824561            57
850   B000QUUFRW        4.809524            84
```

Recommending top 5 products with 100 minimum interactions based on popularity

In [74]:
```
top_n = 5
min_interactions = 100
top_products = get_top_n_products(df_final, top_n, min_rating_count=min_interactions)
print(f"Top {top_n} Products with {min_interactions} minimum interactions based on popularity:\n{top_products}"
```

```
Top 5 Products with 100 minimum interactions based on popularity:
         prod_id  average_rating  rating_count
2316  B003ES5ZUU        4.864130           184
781   B000N99BBC        4.772455           167
2073  B002WE6D44        4.770000           100
4126  B007WTAJTO        4.701220           164
2041  B002V88HFE        4.698113           106
```

We have recommended the **top 5** products by using the popularity recommendation system. Now, let's build a recommendation system using **collaborative filtering.**

Model 2: Collaborative Filtering Recommendation System

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.

In [113…
```
!pip install scikit-surprise
```

```
Defaulting to user installation because normal site-packages is not writeable
Looking in links: /usr/share/pip-wheels
Requirement already satisfied: scikit-surprise in /home/cca15a05-6a3a-4368-9fd9-87213cd22414/.local/lib/python3.11/site-packages (1.1.4)
Requirement already satisfied: joblib>=1.2.0 in /opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-packages (from scikit-surprise) (1.2.0)
Requirement already satisfied: numpy>=1.19.5 in /opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-packages (from scikit-surprise) (1.24.3)
Requirement already satisfied: scipy>=1.6.0 in /opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-packages (from scikit-surprise) (1.11.1)
```

In [20]:
```python
# To compute the accuracy of models
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 tuning model hyperparameters
from surprise.model_selection import GridSearchCV

# For splitting the rating data in train and test datasets
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 K-Fold cross-validation
from surprise.model_selection import KFold

# For implementing clustering-based recommendation system
from surprise import CoClustering
```

Before building the recommendation systems, let's go over some basic terminologies we are going to use:

Relevant item: An item (product in this case) that is actually **rated higher than the threshold rating** is relevant, if the **actual rating is below the threshold then it is a non-relevant item**.

Recommended item: An item that's **predicted rating is higher than the threshold is a recommended item**, if the **predicted rating is below the threshold then that product will not be recommended to the user**.

False Negative (FN): It is the **frequency of relevant items that are not recommended to the user**. If the relevant items are not recommended to the user, then the user might not buy the product/item. This would result in the **loss of opportunity for the service provider**, which they would like to minimize.

False Positive (FP): It is the **frequency of recommended items that are actually not relevant**. In this case, the recommendation system is not doing a good job of finding and recommending the relevant items to the user. This would result in **loss of resources for the service provider**, which they would also like to minimize.

Recall: It is the **fraction of actually relevant items that are recommended to the user**, i.e., if out of 10 relevant products, 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.

While making a recommendation system, it becomes customary to look at the performance of the model. In terms of how many recommendations are relevant and vice-versa, below are some most used performance metrics used in the assessment of recommendation systems.

Precision@k, Recall@ k, and F1-score@k

Precision@k - It is the **fraction of recommended items that are relevant in `top k` predictions**. The 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**.

F1-score@k - It is the **harmonic mean of Precision@k and Recall@k**. When **precision@k and recall@k both seem to be important** then it is useful to use this metric because it is representative of both of them.

Some useful functions

- Below function takes the **recommendation model** as input and gives the **precision@k, recall@k, and F1-score@k** for that model.
- To compute **precision and recall**, **top k** predictions are taken under consideration for each user.
- We will use the precision and recall to compute the F1-score.

In [26]:
```
def precision_recall_at_k(predictions, k=10, threshold=3.5):
    """Return rmse, precision, recall, and F1 score at k metrics for 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()
    f1_scores = 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)  # Sort by estimated rating

        n_rel = sum((true_r >= threshold) for (_, true_r) in user_ratings)
        n_rec_k = sum((est >= threshold) for (est, _) in user_ratings[:k])
        n_rel_and_rec_k = sum(
            ((true_r >= threshold) and (est >= threshold))
            for (est, true_r) in user_ratings[:k]
        )

        # Precision@K
        precision = n_rel_and_rec_k / n_rec_k if n_rec_k != 0 else 0
        precisions[uid] = precision

        # Recall@K
        recall = n_rel_and_rec_k / n_rel if n_rel != 0 else 0
        recalls[uid] = recall

        # F1 Score
        f1_score = (2 * (precision * recall) / (precision + recall)) if (precision + recall) > 0 else 0
        f1_scores[uid] = f1_score

    #return precisions, recalls, f1_scores

    # Mean of all the predicted precisions
    mean_precision  = round((sum(prec for prec in precisions.values()) / len(precisions)), 3)

    # Mean of all the predicted recalls
    mean_recall = round((sum(rec for rec in recalls.values()) / len(recalls)), 3)

    # Mean of all the predicted recalls
    mean_f1_score = round((sum(rec for rec in f1_scores.values()) / len(f1_scores)), 3)

    print('Overall Metrics:')
    rmse = accuracy.rmse(predictions)
    print('Precision: ', mean_precision)
    print('Recall: ', mean_recall)
    print('F1-Score: ', mean_f1_score)

    #return rmse, mean_precision, mean_recall, mean_f1_score
```

Hints:

- To compute **precision and recall**, a **threshold of 3.5 and k value of 10 can be considered for the recommended and relevant ratings**.
- Think about the performance metric to choose.

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`.

In [23]:
```
# Instantiating Reader scale with expected rating scale
reader = Reader(rating_scale=(1, 5))

# Loading the rating dataset
data = Dataset.load_from_df(df_final[['user_id', 'prod_id', 'rating']], reader)

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

Now, we are **ready to build the first baseline similarity-based recommendation system** using the cosine similarity.

Model 2A: User-User Similarity-based Recommendation System

In [56]:
```
# Declaring the similarity options
sim_options = {
    'name': 'cosine',  # or pearson, euclidean, jaccard
    'user_based': True
}

# Initialize the KNNBasic model using sim_options declared, Verbose = False, and setting random_state = 1
knn = KNNBasic(sim_options=sim_options, verbose=False, random_state=1)

# Fit the model on the training data
knn.fit(trainset)

# Make predictions on the test set
predictions = knn.test(testset)

# Let us compute precision@k, recall@k, and f_1 score using the precision_recall_at_k function defined above
precision_recall_at_k(predictions, k=10, threshold=3.5)
```

```
Overall Metrics:
RMSE: 1.0198
Precision:  0.849
Recall:  0.863
F1-Score:  0.835
```

Observations:

- RMSE: The model's predicted ratings are generally 1.017 away from the actual ratings.
- Precision: It is about 85% of the recommended items were relevant to the users. This indicates that the model is effective at identifying and recommending items that users are likely to appreciate. This minimize False Positives (recommended items that are actually not relevant).
- Recall: The model successfully recommended approximately 86% of all relevant items that should have been recommended. This high recall indicates that the model captures most of the relevant items (False Negative)
- F1 Score: This score of 0.83 indicates a strong overall performance, suggesting that the model maintains a good trade-off between precision and recall.

Let's now **predict rating for a user with `userId=A3LDPF5FMB782Z` and `productId=1400501466`** as shown below. Here the user has already interacted or watched the product with productId '1400501466' and given a rating of 5.

In [14]:
```
# Predicting rating for a sample user with an interacted product
user_id = 'A3LDPF5FMB782Z'
product_id = '1400501466'

# Predicting the rating
predicted_rating = knn.predict(user_id, product_id)
print(f'Predicted rating for user {user_id} on product {product_id}: {predicted_rating.est:.2f}')
```

```
Predicted rating for user A3LDPF5FMB782Z on product 1400501466: 4.29
```

Observations:

- The predicted rating of 4.29 is lower than actual rating, suggesting that there may be unknown factors that could lower the accuracy of the model prediction.

In [59]:
```
# Find unique user_id where prod_id is not equal to "1400501466"
unique_users = df_final[df_final['prod_id'] != '1400501466']['user_id'].unique()
print("Count of unique users with product Id not equal to 1400501466: ", len(unique_users))

# Check if user A34BZM6S9L7QI4 is in the list above
print("Is 'A34BZM6S9L7QI4' not related to '1400501466'? ", "Yes" if 'A34BZM6S9L7QI4' in unique_users else "No")
```

```
Count of unique users with product Id not equal to 1400501466:  1540
Is 'A34BZM6S9L7QI4' not related to '1400501466'?  Yes
```

- It can be observed from the above list that **user "A34BZM6S9L7QI4" has not seen the product with productId "1400501466"** as this userId is a part of the above list.

Below we are predicting rating for `userId=A34BZM6S9L7QI4` and `prod_id=1400501466`.

In [16]:
```
# Predicting rating for a sample user with a non interacted product
user_id = 'A34BZM6S9L7QI4'
product_id = '1400501466'
predicted_rating = knn.predict(user_id, product_id)
print(f'Predicted rating for user {user_id} on product {product_id}: {predicted_rating.est:.2f}')
```

```
Predicted rating for user A34BZM6S9L7QI4 on product 1400501466: 3.00
```

Observations:

- A predicted rating of 3.00 is below the mean rating, suggesting that this user may not enjoy the product at all.
- This could mean that the user may tend to rate similar products poorly.
- Similar users may have similar ratings for this product.

Improving similarity-based recommendation system by tuning its hyperparameters

Below, we will be tuning hyperparameters for the `KNNBasic` algorithm. Let's try to understand some of the hyperparameters of the KNNBasic algorithm:

- **k** (int) – The (max) number of neighbors to take into account for aggregation. 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

In [17]:
```
# Setting up parameter grid to tune the hyperparameters
param_grid = {
    'k': [20, 30, 40, 50],  # Different values for k
    'min_k': [1, 5, 10],        # Different values for min_k
    'sim_options': {
        'name': ['cosine', 'msd', 'pearson', 'pearson_baseline'],  # Similarity measures
        'user_based': [True]
    }
}

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

# Fitting the data
gs.fit(data)

# Best RMSE score
print("Best RMSE: ", gs.best_score)

# Combination of parameters that gave the best RMSE score
best_params = gs.best_params['rmse']
print("Best parameters for RMSE: ", best_params)
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
```

```
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
```

```
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
```

```
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
```

```
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Computing the pearson similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
Best RMSE:  {'rmse': 0.9704659623659179}
Best parameters for RMSE:  {'k': 50, 'min_k': 5, 'sim_options': {'name': 'cosine', 'user_based': True}}
```

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

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

In [45]:
```
# Using the optimal similarity measure for user-user based collaborative filtering
k = 50
min_k = 5
sim_options = {
    'name': 'cosine',
    'user_based': True  # User-user similarity
}

# Creating an instance of KNNBasic with optimal hyperparameter values
sim_user_user_optimized = KNNBasic(k=k, min_k=min_k, sim_options=sim_options)

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

# Make predictions on the test set
predictions = sim_user_user_optimized.test(testset)

# Let us compute precision@k and recall@k also with k =10
precision_recall_at_k(predictions, k=10, threshold=3.5)
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
Overall Metrics:
RMSE: 0.9588
Precision:  0.843
Recall:  0.898
F1-Score:  0.848
```

Observations:

- The RMSE of the optimized model is 0.9588, which is lower than that of the baseline model (1.0172). This indicates that the tuned model makes predictions that are closer to the actual ratings, which is a improvement in prediction accuracy.
- The precision 0.843 is slightly lower than that of the baseline model (0.855). This suggests that the tuned model is recommending a marginally higher proportion of relevant items among its top recommendations, but the difference is not significant.
- The recall 0.898 is higher than that of the baseline model (0.859). This indicates that the tuned model is better at capturing relevant items, successfully identifying more of the items that users would likely find valuable.

- The F1 score 0.848 is higher than that of the baseline model (0.835). This suggests that the tuned model achieves a better balance between precision and recall, making it more effective overall in recommending relevant items.

Steps:

- **Predict rating for the user with `userId="A3LDPF5FMB782Z"` , and `prod_id= "1400501466"` using the optimized model**
- **Predict rating for `userId="A34BZM6S9L7QI4"` who has not interacted with `prod_id ="1400501466"` , by using the optimized model**
- **Compare the output with the output from the baseline model**

In [46]:
```
# Use sim_user_user_optimized model to recommend for userId "A3LDPF5FMB782Z" and productId 1400501466
user_id = "A3LDPF5FMB782Z"
prod_id = "1400501466"

# Make a prediction for the given user and product
predicted_rating = sim_user_user_optimized.predict(user_id, prod_id).est

# Display the predicted rating
print(f"Predicted rating for user {user_id} on product {prod_id}: {predicted_rating:.2f}")
```

```
Predicted rating for user A3LDPF5FMB782Z on product 1400501466: 3.80
```

In [47]:
```
# Use sim_user_user_optimized model to recommend for userId "A34BZM6S9L7QI4" and productId "1400501466"
user_id = "A34BZM6S9L7QI4"
prod_id = "1400501466"

# Predict the rating
predicted_rating = sim_user_user_optimized.predict(user_id, prod_id).est

# Output the predicted rating
print(f"Predicted rating for User {user_id} for Product {prod_id}: {predicted_rating:.2f}")
```

```
Predicted rating for User A34BZM6S9L7QI4 for Product 1400501466: 4.30
```

Observations:

- 1. User A3LDPF5FMB782Z
  - The predicted rating for this user decreased from 4.29 to 3.80 after hyperparameter tuning.
  - This could indicate that the improved model is better at accounting for the user's interaction history or the similarity of other users in the dataset that affects the prediction in a way that balances overall user preferences.
- 2. User A34BZM6S9L7QI4
  - The predicted rating for this user increased from 3.00 to 4.30 after tuning the model.
  - This significant increase suggests that the improved model has likely identified relevant similarities with other users that the baseline model did not recognize. A predicted rating of 3.00 indicates a low expectation of relevance in the baseline, while the improved model reflects a much more favorable view of the product for this user.

Identifying similar users to a given user (nearest neighbors)

We can also find out **similar users to a given user** or its **nearest neighbors** based on this KNNBasic algorithm. Below, we are finding the 5 most similar users to the first user in the list with internal id 0, based on the `msd` distance metric.

In [ ]:
```
# 0 is the inner id of the above user
sim_options = {
    'name': 'msd',  # Mean Squared Difference
    'user_based': True  # User-user similarity
}

# Create and fit the model on the training set
model = KNNBasic(sim_options=sim_options)
model.fit(trainset)

# Function to get similar users
def get_similar_users(model, inner_uid, n=5):
    # Get nearest neighbors
    similar_users = model.get_neighbors(inner_uid, k=n)

    # Convert inner IDs back to raw user IDs
    similar_users_raw = [model.trainset.to_raw_uid(iid) for iid in similar_users]

    return similar_users_raw

# Specify the inner user ID (0 in this case)
inner_user_id = 0  # Inner ID of the user

# Get similar users
similar_users = get_similar_users(model, inner_user_id, n=5)
print(f"Similar users to User with inner ID {inner_user_id}: {similar_users}")
```

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: A user id **against which we want the recommendations**
- top_n: The **number of products we want to recommend**
- algo: the algorithm we want to use **for predicting the ratings**
- The output of the function is a **set of top_n items** recommended for the given user_id based on the given algorithm

In [41]:
```
# Define the recommendation function
def get_recommendations(data, user_id, top_n, algo):

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

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

    # Extracting those product ids which the user_id has not interacted yet
    non_interacted_products = user_item_interactions_matrix.loc[user_id][user_item_interactions_matrix.loc[user_

    # Looping through each of the product ids which user_id has not interacted yet
    for item_id in non_interacted_products:

        # Predicting the ratings for those non interacted product 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 products for this user
```

Predicting top 5 products for userId = "A3LDPF5FMB782Z" with similarity based recommendation system

In [42]:
```
# Making top 5 recommendations for user_id "A3LDPF5FMB782Z" with a similarity-based recommendation engine
# Create and fit the KNNBasic model
algo = KNNBasic(sim_options={'name': 'cosine', 'user_based': True})
algo.fit(trainset)

# Make recommendations"
user_id = "A3LDPF5FMB782Z"
top_n = 5
recommended_items = get_recommendations(df, user_id, top_n, algo)
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
```

In [43]:
```
# Building the dataframe for above recommendations with columns "prod_id" and "predicted_ratings"
recommended_df = pd.DataFrame(recommended_items, columns=['prod_id', 'predicted_rating'])

# Display the recommendations DataFrame
print(f"Top 5 Recommendations for User ID {user_id}:")
print(recommended_df)
```

```
Top 5 Recommendations for User ID A3LDPF5FMB782Z:
      prod_id  predicted_rating
0  1400599997                 5
1  9983891212                 5
2  B00001P4XA                 5
3  B00001W0DI                 5
4  B00002EQCW                 5
```

Model 2B: Item-Item Similarity-based Collaborative Filtering Recommendation System

- Above we have seen **similarity-based collaborative filtering** where similarity is calculated **between users**. Now let us look into similarity-based collaborative filtering where similarity is seen **between items**.

In [27]:
```
# Declaring the similarity options
similarity_options = {
    'name': 'cosine',        # Choose similarity type: 'cosine', 'pearson', or 'msd'
    'user_based': False,     # Set user_based to False for item-based filtering
    'min_support': 3         # Minimum common ratings for similarity
}
```

```
# KNN algorithm is used to find desired similar items. Use random_state=1
# trainset, testset = train_test_split(data, test_size=0.2, random_state=1)
model = KNNBasic(sim_options=similarity_options)

# Train the algorithm on the trainset, and predict ratings for the test set
model.fit(trainset)

# Make predictions on the test set
predictions = model.test(testset)

# Let us compute precision@k, recall@k, and f_1 score with k = 10
precision_recall_at_k(predictions, k=10, threshold=3.5)
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
Overall Metrics:
RMSE: 1.0344
Precision:  0.831
Recall:  0.882
F1-Score:  0.834
```

Observations:

- RMSE of 1.03 indicates the average deviation of the predicted ratings from the actual ratings. A lower RMSE suggests better model performance.
- A precision of 0.832 means that approximately 82.6% of the items recommended to users were relevant. This shows that the model is good at recommending relevant items.
- A recall of 0.878 indicates that the model was able to identify about 87.9% of the relevant items available to users. High recall is beneficial, especially in recommendation systems.
- An F1 score of 0.833 is a harmonic mean of precision and recall, indicating a good balance between the two. Higher F1 scores suggest better overall performance of the model.

Let's now **predict a rating for a user with `userId = A3LDPF5FMB782Z` and `prod_Id = 1400501466`** as shown below. Here the user has already interacted or watched the product with productId "1400501466".

In [48]:
```
# Predicting rating for a sample user with an interacted product
user_id = 'A3LDPF5FMB782Z'
product_id = '1400501466'

# Predict the rating for the user and product
predicted_rating = model.predict(user_id, product_id)

print(f'Predicted rating for user {user_id} and product {product_id}: {predicted_rating.est:.2f}')
```

```
Predicted rating for user A3LDPF5FMB782Z and product 1400501466: 5.00
```

Observations:

- A predicted rating of 5.00 indicates that the user is expected to rate this product at the maximum level based on their previous interactions and the behavior of similar users.

Below we are **predicting rating for the `userId = A34BZM6S9L7QI4` and `prod_id = 1400501466`**.

In [28]:
```
# Predicting rating for a sample user with a non interacted product
new_user_id = 'A34BZM6S9L7QI4'
new_product_id = '1400501466'

#Verify if A34BZM6S9L7QI4 is related to 1400501466
user_product_relation = df_final[
    (df_final['user_id'] == new_user_id) &
    (df_final['prod_id'] == new_product_id)
]

print('Result:')
print(f'User {new_user_id} has {"interacted with" if not user_product_relation.empty else "NOT interacted with"}

predicted_rating_new = model.predict(new_user_id, new_product_id)

# Output the predicted rating
print(f'Predicted rating for user {new_user_id} and product {new_product_id}: {predicted_rating_new.est:.2f}')
```

```
Result:
User A34BZM6S9L7QI4 has NOT interacted with product 1400501466.
Predicted rating for user A34BZM6S9L7QI4 and product 1400501466: 4.30
```

Observations:

- Despite the lack of past interaction, a predicted rating of 4.30 suggests that the model believes this user would likely rate the product positively.
- This may be because of the ratings of similar users who have interacted with the product.

- The product might have features that match the user's preferences based on previous ratings of other products.

Hyperparameter tuning the item-item similarity-based model

- Use the following values for the param_grid and tune the model.
  - 'k': [10, 20, 30]
  - 'min_k': [3, 6, 9]
  - 'sim_options': {'name': ['msd', 'cosine']
  - 'user_based': [False]
- Use GridSearchCV() to tune the model using the 'rmse' measure
- Print the best score and best parameters

In [24]:
```
# Setting up parameter grid to tune the hyperparameters
param_grid = {
    'k': [10, 20, 30],
    'min_k': [3, 6, 9],
    'sim_options': {
        'name': ['msd', 'cosine'],
        'user_based': [False]  # Item-based
    }
}

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

#reader = Reader(rating_scale=(1, 5))
#data = Dataset.load_from_df(df_final[['user_id', 'prod_id', 'rating']], reader)

# Fitting the data
grid_search.fit(data)

# Find the best RMSE score
print("Best RMSE Score: ", grid_search.best_score['rmse'])

# Find the combination of parameters that gave the best RMSE score
print("Best Parameters: ", grid_search.best_params['rmse'])
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
```

```
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Best RMSE Score:  0.9750984762383759
Best Parameters:  {'k': 20, '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.**

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

Use the best parameters from GridSearchCV to build the optimized item-item similarity-based model. Compare the performance of the optimized model with the baseline model.

In [29]:

```
# Using the optimal similarity measure for item-item based collaborative filtering
# Creating an instance of KNNBasic with optimal hyperparameter values
optimized_model = KNNBasic(k=20, min_k=6, sim_options={'name': 'msd', 'user_based': False})

# Training the algorithm on the trainset
optimized_model.fit(trainset)
```

```
# Make predictions on the test set
predictions = optimized_model.test(testset)

# Let us compute precision@k and recall@k, f1_score with k=10
precision_recall_at_k(predictions, k=10, threshold=3.5)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Overall Metrics:
RMSE: 0.9687
Precision:  0.832
Recall:  0.89
F1-Score:  0.838
```

Observations (compated with Baseline Model):

- RMSE: The optimized model has a significantly lower RMSE, indicating improved accuracy in predicting user ratings
- Precision: The optimized model has a slight improvement in precision. This means that the proportion of relevant items among the recommended items is maintained, indicating that the model continues to provide relevant recommendations.
- Recall: Improvement in recall indicates that the optimized model is better at identifying relevant items among all actual relevant items. This suggests that the optimized model retrieves more relevant recommendations compared to the baseline.
- F1-score: The higher value of F1-score indicates that the optimized model is more effective in providing relevant recommendations.

Steps:

- **Predict rating for the user with `userId="A3LDPF5FMB782Z"`, and `prod_id= "1400501466"` using the optimized model**
- **Predict rating for `userId="A34BZM6S9L7QI4"` who has not interacted with `prod_id ="1400501466"`, by using the optimized model**
- **Compare the output with the output from the baseline model**

In [49]:
```
# Use sim_item_item_optimized model to recommend for userId "A3LDPF5FMB782Z" and productId "1400501466"
# User and product IDs
user_id = "A3LDPF5FMB782Z"
product_id = "1400501466"

# Predict the rating using the optimized model
predicted_rating = optimized_model.predict(user_id, product_id)

# Display the predicted rating
rating_count = len(df_final[(df_final['user_id'] == user_id) & (df_final['prod_id'] == product_id)])
print(f'Ratings count of product {product_id} by user {user_id}: {rating_count}')
print(f"Predicted rating for user {user_id} on product {product_id}: {predicted_rating.est:.2f}")
```

```
Ratings count of product 1400501466 by user A3LDPF5FMB782Z: 1
Predicted rating for user A3LDPF5FMB782Z on product 1400501466: 4.67
```

In [50]:
```
# Use sim_item_item_optimized model to recommend for userId "A34BZM6S9L7QI4" and productId "1400501466"
# User and product IDs
user_id = "A34BZM6S9L7QI4"
product_id = "1400501466"

# Predict the rating using the optimized model
predicted_rating = optimized_model.predict(user_id, product_id)

# Display the predicted rating
rating_count = len(df_final[(df_final['user_id'] == user_id) & (df_final['prod_id'] == product_id)])
print(f'Ratings count of product {product_id} by user {user_id}: {rating_count}')
print(f"Predicted rating for user {user_id} on product {product_id}: {predicted_rating.est:.2f}")
```

```
Ratings count of product 1400501466 by user A34BZM6S9L7QI4: 0
Predicted rating for user A34BZM6S9L7QI4 on product 1400501466: 4.30
```

Observations:

- Rating of 4.82 means that the user A3LDPF5FMB782Z has good experience of product 1400501466.
- User A34BZM6S9L7QI4 did not ever rate the product 1400501466 but the model predicts a positive rating of 4.29, suggesting that the model's pediction is based on its prediction on similarities with other users or items.

Identifying similar items to a given item (nearest neighbors)

We can also find out **similar items** to a given item or its nearest neighbors based on this **KNNBasic algorithm**. Below we are finding the 5 most similar items to the item with internal id 0 based on the `msd` distance metric.

In [52]:
```
# internal id of item
internal_id=0

# Define the KNNBasic algorithm with msd distance metric
sim_options = {
```

```
    'name': 'msd',
    'user_based': False  # Set to True for user-based collaborative filtering
}
knn_model = KNNBasic(sim_options=sim_options)
knn_model.fit(trainset)

similar_items = knn_model.get_neighbors(internal_id, k=5)
print(f"5 most similar items similar to item of internal id {internal_id}: {similar_items}")
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
5 most similar items similar to item of internal id 0: [2, 63, 93, 213, 216]
```

Predicting top 5 products for userId = "A1A5KUIIIHFF4U" with similarity based recommendation system.

Hint: Use the get_recommendations() function.

In [53]:
```
# Making top 5 recommendations for user_id A1A5KUIIIHFF4U with similarity-based recommendation engine.
user_id="A1A5KUIIIHFF4U"

recommendations = get_recommendations(df_final, user_id, 5, knn_model)
```

In [54]:
```
# Building the dataframe for above recommendations with columns "prod_id" and "predicted_ratings"
recommendations_df = pd.DataFrame(recommendations, columns=['prod_id', 'predicted_ratings'])
print(recommendations_df)
```

```
      prod_id  predicted_ratings
0  9983891212                  5
1  B00000DM9W                  5
2  B00000J1V5                  5
3  B00000K135                  5
4  B00000K4KH                  5
```

Now as we have seen **similarity-based collaborative filtering algorithms**, let us now get into **model-based collaborative filtering algorithms**.

Model 3: Model-Based Collaborative Filtering - Matrix Factorization

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.

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**.

In [30]:
```
# Using SVD matrix factorization. Use random_state = 1
baseline_svd_model = SVD(random_state=1)

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

# Make predictions on the test set
predictions = baseline_svd_model.test(testset)

# Use the function precision_recall_at_k to compute precision@k, recall@k, F1-Score, and RMSE
precision_recall_at_k(predictions, k=10, threshold=3.5)
```

```
Overall Metrics:
RMSE: 0.8994
Precision:  0.843
Recall:  0.882
F1-Score:  0.842
```

Observations:

- RMSE: The model's predictions deviate from the actual ratings by about 0.90 points.
- Precision: About 84.4% of the recommended items were relevant
- Recall: The model successfully identified 88% of all relevant items within the dataset
- F1-Score: An F1-Score of 0.841 indicates that the model has a good balance between precision and recall

Let's now predict the rating for a user with `userId = "A3LDPF5FMB782Z"` and `prod_id = "1400501466`.

In [33]:
```
# Making prediction
# Predicting using the SVD model
user_id = "A3LDPF5FMB782Z"
prod_id = "1400501466"
```

```
# Make a prediction for the given user and product using SVD
predicted_rating_svd = baseline_svd_model.predict(user_id, prod_id).est

# Display the predicted rating
print(f"Predicted rating for user {user_id} on product {prod_id} using SVD: {predicted_rating_svd:.2f}")
```

```
Predicted rating for user A3LDPF5FMB782Z on product 1400501466 using SVD: 4.32
```

Observations:

- The predicted rating of 4.32 is comparable to actual rating of 5. This indicates that the SVD model effectively recognizes this user's strong preference for the product.

Below we are predicting rating for the `userId = "A34BZM6S9L7QI4"` and `productId = "1400501466"`.

In [35]:
```
# Making prediction
# Predicting using the SVD model
user_id = "A34BZM6S9L7QI4"
prod_id = "1400501466"

# Make a prediction for the given user and product using SVD
predicted_rating_svd = baseline_svd_model.predict(user_id, prod_id).est

# Display the predicted rating
print(f"Predicted rating for user {user_id} on product {prod_id} using SVD: {predicted_rating_svd:.2f}")
```

```
Predicted rating for user A34BZM6S9L7QI4 on product 1400501466 using SVD: 4.59
```

Observations:

- The predicted rating of 4.59 indicates that the product may be liked by the User A34BZM6S9L7QI4, even there is no related interaction.
- This relatively high prediction may suggest that the model finds similarities between this user and others who have rated the product favorably or that the characteristics of the product align well with the preferences inferred from this user's behavior and rating history.

Improving Matrix Factorization based recommendation system by tuning its hyperparameters

Below we will be tuning only three hyperparameters:

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

In [31]:
```
# Set the parameter space to tune
param_grid = {
    'n_epochs': [5, 10, 15, 20],
    'lr_all': [0.002, 0.005, 0.01, 0.02],
    'reg_all': [0.2, 0.4, 0.6, 0.8]
}

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

# Fitting data
grid_search.fit(data)

# Best RMSE score
best_rmse = grid_search.best_score['rmse']
print(f"Best RMSE: {best_rmse:.4f}")

# Combination of parameters that gave the best RMSE score
best_params = grid_search.best_params['rmse']
print(f"Best parameters: {best_params}")
```

```
Best RMSE: 0.8986
Best parameters: {'n_epochs': 20, 'lr_all': 0.01, 'reg_all': 0.2}
```

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

In [36]:
```
# Build the optimized SVD model using optimal hyperparameter search. Use random_state=1
svd_algo_optimized = SVD(n_epochs=best_params['n_epochs'],
                         lr_all=best_params['lr_all'],
                         reg_all=best_params['reg_all'],
                         random_state=1)

# Train the algorithm on the trainset
svd_algo_optimized.fit(trainset)
```

```
# Optionally evaluate the model on the test set
predictions = svd_algo_optimized.test(testset)

# Use the function precision_recall_at_k to compute precision@k, recall@k, F1-Score, and RMSE
precision_recall_at_k(predictions, k=10, threshold=3.5)
```

```
Overall Metrics:
RMSE: 0.8902
Precision:  0.845
Recall:  0.882
F1-Score:  0.844
```

Observations (compared with baseline model):

- The SVD optimized model shows very small improvements in RMSE, precision, and F1-Score, indicating better predictive accuracy and relevance of recommendations.
- The small improvements implies that hyperparameter tuning might require substantial effort for further optimization.

Steps:

- **Predict rating for the user with `userId="A3LDPF5FMB782Z"`, and `prod_id= "1400501466"` using the optimized model**
- **Predict rating for `userId="A34BZM6S9L7QI4"` who has not interacted with `prod_id ="1400501466"`, by using the optimized model**
- **Compare the output with the output from the baseline model**

In [37]:
```
# Use svd_algo_optimized model to recommend for userId "A3LDPF5FMB782Z" and productId "1400501466"
user_id = "A3LDPF5FMB782Z"
prod_id = "1400501466"

# Predict the rating
predicted_rating = svd_algo_optimized.predict(user_id, prod_id).est

# Display the predicted rating
print(f"Predicted rating for user {user_id} on product {prod_id}: {predicted_rating:.2f}")
```

```
Predicted rating for user A3LDPF5FMB782Z on product 1400501466: 4.33
```

In [38]:
```
# Use svd_algo_optimized model to recommend for userId "A34BZM6S9L7QI4" and productId "1400501466"
user_id = "A34BZM6S9L7QI4"
prod_id = "1400501466"

# Predict the rating
predicted_rating = svd_algo_optimized.predict(user_id, prod_id).est

# Display the predicted rating
print(f"Predicted rating for user {user_id} on product {prod_id}: {predicted_rating:.2f}")
```

```
Predicted rating for user A34BZM6S9L7QI4 on product 1400501466: 4.43
```

Conclusion:

Model Performance:

- Improved SVD Model exhibits the best overall performance with the lowest RMSE (0.8902) among the models, indicating better predictive accuracy compared to the other collaborative filtering methods and the rank-based system.
- SVD performs slightly better in balancing precision and recall, closely matching the other collaborative filtering models. This indicates that it not only predicts ratings well but also identifies relevant products effectively.

Predictions:

- For user A3LDPF5FMB782Z, who has an existing rating of 5 for product 1400501466 (around 4.29 to 5.00 for the product). The predictions suggest that the models can give accurate prediction for existing users predictions.
- For user A34BZM6S9L7QI4 who has not rated the product, the predictions vary across models but indicate a reasonably positive predicted interest. It may indicate that those models predict based on similar products.

Recommendations:

- Hyperparameters tuning can lead to better predictions but may not always significantly enhance metrics like precision and recall. It's worth evaluating the trade-off based on the context of the application.
- The Improved SVD Model is recommended as the best-performing model due to its low RMSE, balanced precision and recall, and strong predictive accuracy. It shows the most promise in delivering personalized recommendations based on user preferences while managing to understand user-item relations effectively. It ultimately reflects a more robust model suitable for real-world applications, where understanding nuances in user behavior is essential.