Pricilla Nakyazze Project 2 revised Submission

June 19, 2025

[8]:
```
import pandas as pd
import importlib
```

[9]:
```
import numpy as np
```

[10]:
```
import os
```

[11]:
```
file_path = "C:/Users/pricc/Downloads/goodreads_interactions copy.csv.xlsx"
```

[12]:
```
df = pd.read_excel("goodreads_interactions copy.csv.xlsx")
```

[13]:
```
print(df.head())
```

```
   user_id  book_id  is_read  rating  is_reviewed
0      255      948        1       5            0
1      255      947        1       5            1
2      255      946        1       5            0
3      255      945        1       5            0
4      255      944        1       5            0
```

This Project explores three different approaches to building a book recommendation system, the datais from the Goodreads dataset. The objective is to evaluate how different recommendation algorithms wor in practice, compare their effectiveness, and understand their relative strengths and weaknesse s2. Overview of the Three Approaches 1. Content-Based Filtering (CBF): Recommends books based on user preferences and item characteris ics inferred from ratings. 2. Item-Item Collaborative Filtering: Recommends books that are rated similarly by users. 3. User-User Collaborative Filtering: Recommends books by identifying users with similar tastes.

1 The code below tackles Content-Based Filtering Normalization and recommends books that are similar in rating pattern.

The dataset contains user_id, book_id, and rating. Cosine similarity and standard rating normalization were used across models. Cosine similarity is the cosine of the angle between the vectors; that is, it is the dot product of the vectors divided by the product of their lengths. .

[24]:
```
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import StandardScaler

# Load data (already loaded in variable df)
```

```python
# Filter for books the user has rated (is_read == 1 and rating > 0)
interactions = df[df['is_read'] == 1]

# Pivot to create user-item matrix (rows = users, columns = books, values =
↪rating)
user_item_matrix = interactions.pivot_table(index='user_id', columns='book_id',
↪values='rating').fillna(0)

# Transpose to get item-user matrix for similarity (rows = books, columns =
↪users)
item_user_matrix = user_item_matrix.T

# Standardize ratings (optional but improves cosine similarity results)
scaler = StandardScaler()
item_user_scaled = scaler.fit_transform(item_user_matrix)

# Compute cosine similarity between books
book_similarity = cosine_similarity(item_user_scaled)

# Put it into a DataFrame for easy lookup
book_similarity_df = pd.DataFrame(book_similarity, index=item_user_matrix.
↪index, columns=item_user_matrix.index)

# --- Function to recommend similar books ---
def recommend_books(book_id, n=5):
    if book_id not in book_similarity_df.columns:
        return f"Book ID {book_id} not found in similarity matrix."
    similar_books = book_similarity_df[book_id].sort_values(ascending=False)
    return similar_books.iloc[1:n+1]  # Skip the book itself (similarity = 1)

# Example: Recommend 5 books similar to book ID 948
recommendations = recommend_books(948, n=5)
print("Books similar to book 948 rcommended by CBF :\n", recommendations)
```

```
Books similar to book 948 rcommended by CBF :
 book_id
945    1.0
938    1.0
939    1.0
940    1.0
941    1.0
Name: 948, dtype: float64
```

The book IDs above have been read,

Have near-identical rating vectors,

Are top matches by cosine similarity (1.0)

Fit the CBF model based on implicit collaborative item profiles. The value of 1.0 indicates a perfect similarity score or the highest possible recommendation strength based on the CBF algorithm's calculation.

The Pros of content Based Filtering that make it a practical choice is there is no need for metadata hence very easy to interpret

CBF Doesn't use semantic info (e.g., genres, descriptions). For example my CBF recommender is based on available user rating alone and what was rated highly (5).

It is Personalized because it's based on actual rating, if a user gave high ratings to a group of books, CBF can find other books with similar user rating patterns when making recommendations.

Cons: Recommends only similar items in this case a similar rating this limits diversity.

Cosine similarity may overestimate weak similarities. CBF requires enough interactions per item for stable comparison

2 Item-Item Collaborative Filtering

item-to-item Collaborative Filtering, is a form of collaborative filtering for recommender systems based on the similarity between items calculated using people's ratings of those items.

Lets find Books with a similar rating to book ID 948 using Item to Item Collaborative Filtering

[25]:
```
# Create user-item rating matrix
user_item_matrix = df.pivot_table(index='user_id', columns='book_id',␣
↪values='rating').fillna(0)

# Transpose to get items as rows
item_user_matrix = user_item_matrix.T

# Compute cosine similarity between items (books)
item_similarity = cosine_similarity(item_user_matrix)

# Create a DataFrame for easy lookup
item_similarity_df = pd.DataFrame(item_similarity, index=item_user_matrix.
↪index, columns=item_user_matrix.index)

# --- Recommend similar items function ---
def recommend_similar_items(book_id, n=5):
    if book_id not in item_similarity_df.columns:
        return f"Book ID {book_id} not found in similarity matrix."
    # Get top n similar items, skip the item itself
    similar_books = item_similarity_df[book_id].sort_values(ascending=False).
↪iloc[1:n+1]
    return similar_books

#  Example: Recommend books similar to book ID 948
recommendations = recommend_similar_items(948, n=5)
```

```
print("Books similar to book 948 based on how users have rated them.:\n",␣
↪recommendations)
```

```
Books similar to book 948 based on how users have rated them.:
 book_id
948    1.0
933    1.0
934    1.0
935    1.0
936    1.0
Name: 948, dtype: float64
```

Books 933, 934, 935, and 936 are the most similar to book 948. Each book has a cosine similarity score of 1.0 with book 948.

Some Item-Item Collaborative Filtering cons are :

With item to Item Collaborative Filtering in this case new books with few ratings can't be recommended effectively, multiple users need to have rated both items to compute similarity and two books may be similar in rating patterns but unrelated in theme or genre.

A few Pros for Item-Item Collaborative Filtering are:

Item Collaborative Filtering works well when many users rate items so even with sparse user data, items may still have lots of ratings. Similarities are stable because Item preferences are less volatile than user behavior. books don't change.

There is no need for item metadata because recommendations are based purely on user interactions.

Evaluation

[27]:
```
pip install pandas scikit-learn openpyxl
```

```
Defaulting to user installation because normal site-packages is not writeable
Looking in links: /usr/share/pip-wheels
Requirement already satisfied: pandas in /opt/conda/envs/anaconda-
panel-2023.05-py310/lib/python3.11/site-packages (2.0.3)
Requirement already satisfied: scikit-learn in /opt/conda/envs/anaconda-
panel-2023.05-py310/lib/python3.11/site-packages (1.3.0)
Requirement already satisfied: openpyxl in /opt/conda/envs/anaconda-
panel-2023.05-py310/lib/python3.11/site-packages (3.0.10)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/conda/envs/anaconda-panel-2023.05-py310/lib/python3.11/site-packages (from
pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/envs/anaconda-
panel-2023.05-py310/lib/python3.11/site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in /opt/conda/envs/anaconda-
panel-2023.05-py310/lib/python3.11/site-packages (from pandas) (2023.3)
Requirement already satisfied: numpy>=1.21.0 in /opt/conda/envs/anaconda-
panel-2023.05-py310/lib/python3.11/site-packages (from pandas) (1.24.3)
Requirement already satisfied: scipy>=1.5.0 in /opt/conda/envs/anaconda-
```

```
panel-2023.05-py310/lib/python3.11/site-packages (from scikit-learn) (1.11.1)
Requirement already satisfied: joblib>=1.1.1 in /opt/conda/envs/anaconda-
panel-2023.05-py310/lib/python3.11/site-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /opt/conda/envs/anaconda-
panel-2023.05-py310/lib/python3.11/site-packages (from scikit-learn) (2.2.0)
Requirement already satisfied: et_xmlfile in /opt/conda/envs/anaconda-
panel-2023.05-py310/lib/python3.11/site-packages (from openpyxl) (1.1.0)
Requirement already satisfied: six>=1.5 in /opt/conda/envs/anaconda-
panel-2023.05-py310/lib/python3.11/site-packages (from python-
dateutil>=2.8.2->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

Build Item Profile to represent the characteristics of each item, enabling the system to understand and match items with user preferences.

[28]:
```python
from sklearn.model_selection import train_test_split

# Make sure your dataset has 'user_id', 'book_id', 'rating'
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
```

[29]:
```python
train_matrix = train_df.pivot(index='user_id', columns='book_id',␣
↪values='rating')
```

[30]:
```python
# Build item profiles as normalized rating vectors (item-user matrix)
item_profiles = train_matrix.T.fillna(0)
item_profiles_norm = item_profiles.div(np.linalg.norm(item_profiles, axis=1),␣
↪axis=0)  # cosine norm
print(item_profiles_norm.head())
```

```
user_id  2    3    4    5    34   55   80   84   90   99   … \
book_id                                                     …
43        0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …
134       0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …
233       0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …
234       0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …
332       0.0  0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  …

user_id  818  864  888  1029 1138 1225 1518 1685 1945 2081
book_id
43        0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
134       0.0  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
233       0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
234       0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
332       0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

[5 rows x 44 columns]
```

Build a similarity df to identify and quantify the relationships between users or items within the system.

[31]:
```
# Replace NaNs with 0
item_profiles_clean = item_profiles_norm.fillna(0)

# Now compute similarity
item_similarity = cosine_similarity(item_profiles_clean)

# Build a similarity DataFrame
item_similarity_df = pd.DataFrame(item_similarity, index=item_profiles.index,␣
↪columns=item_profiles.index)
```

[32]:
```
print(item_similarity_df.head())
```

```
book_id  43      134     233     234     332     334     384     412     \
book_id
43          1.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
134         0.0     1.0     0.0     0.0     0.0     0.0     0.0     0.0
233         0.0     0.0     1.0     1.0     1.0     0.0     0.0     0.0
234         0.0     0.0     1.0     1.0     1.0     0.0     0.0     0.0
332         0.0     0.0     1.0     1.0     1.0     0.0     0.0     0.0

book_id  446     460     …  134798  134799  134800  134802  134803  134804  \
book_id                  …
43          0.0     0.0  …     0.0     0.0     0.0     0.0     0.0     0.0
134         0.0     0.0  …     0.0     0.0     0.0     0.0     0.0     0.0
233         0.0     0.0  …     0.0     0.0     0.0     0.0     0.0     0.0
234         0.0     0.0  …     0.0     0.0     0.0     0.0     0.0     0.0
332         0.0     0.0  …     0.0     0.0     0.0     0.0     0.0     0.0

book_id  134805  134806  135519  164541
book_id
43          0.0     0.0     1.0     0.0
134         0.0     0.0     0.0     1.0
233         0.0     0.0     0.0     0.0
234         0.0     0.0     0.0     0.0
332         0.0     0.0     0.0     0.0

[5 rows x 476 columns]
```

Books with book_id 233, 234, and 332 have a similarity value of 1.0 with each other, suggesting they are very similar. The 0.0 values between most other book pairs indicate that the books are not considered similar based on the similarity calculation.

3 User-Based Collaborative Filtering

A technique used to predict the items that a user might like on the basis of ratings given to that item by other users who have similar taste with that of the target user.

Create a user-item interaction matrix in the context of user-based collaborative filtering for the book recommendation system.

[33]:

```python
from sklearn.metrics import mean_squared_error


# Step 1: Load data
ratings_df = pd.read_excel("goodreads_interactions copy.csv.xlsx")

# Step 2: Create ratings matrix
ratings_matrix = ratings_df.pivot_table(index='user_id', columns='book_id',␣
↪values='rating')
train_matrix = ratings_matrix.copy()


# Step 3: Normalize
def normalize(matrix):
    means = matrix.mean(axis=1)
    norm_matrix = (matrix.T - means).T
    return norm_matrix.fillna(0), means

train_norm, user_means = normalize(train_matrix)

# Step 4: Compute similarities
def compute_similarity(matrix):
    return cosine_similarity(matrix)

user_similarity = compute_similarity(train_norm)
item_similarity = compute_similarity(train_norm.T)

# Step 5: Predict ratings
def predict_ratings(similarity, ratings, type='user', means=None):
    eps = 1e-8
    if type == 'user':
        weighted_sum = similarity.dot(ratings)
        sim_sums = np.abs(similarity).sum(axis=1).reshape(-1, 1) + eps
        pred = weighted_sum / sim_sums
        if means is not None:
            pred = pred + means.values.reshape(-1, 1)
        return pd.DataFrame(pred, index=ratings.index, columns=ratings.columns)
    elif type == 'item':
        weighted_sum = ratings.dot(similarity)
        sim_sums = np.abs(similarity).sum(axis=1).reshape(1, -1) + eps
        pred = weighted_sum / sim_sums
        return pd.DataFrame(pred, index=ratings.index, columns=ratings.columns)

user_preds = predict_ratings(user_similarity, train_norm, type='user',␣
↪means=user_means)
item_preds = predict_ratings(item_similarity, train_matrix, type='item')
```

[108]:
```python
print(user_preds.head())
```

```
book_id      43        134       233       234       332       334       384     \
user_id
2         3.320755  3.320755  3.320755  3.320755  3.320755  3.320755  3.320755
3         1.000000  1.000000  1.000000  1.000000  1.000000  3.890740  1.000000
4         3.546180  3.541667  3.998319  3.001986  3.998319  3.541667  3.541667
5         2.208333  2.208333  2.208333  2.208333  2.208333  2.208333  2.208333
34        4.000000  4.000000  4.000000  4.000000  4.000000  4.000000  4.000000

book_id      412       439       446     …    134799    134800    134801  \
user_id                                  …
2         3.320755  3.320755  3.320755  …  3.320755  3.320755  3.320755
3         0.036420  1.000000  1.000000  …  1.000000  1.000000  1.000000
4         3.541667  3.541667  3.541667  …  3.541667  3.541667  3.541667
5         2.208333  2.208333  2.208333  …  2.208333  2.208333  2.208333
34        4.000000  4.000000  4.000000  …  4.000000  4.000000  4.000000

book_id     134802    134803    134804    134805    134806    135519    164541
user_id
2         3.320755  3.320755  3.320755  3.320755  3.320755  3.320755  3.320755
3         1.000000  1.000000  1.000000  1.000000  1.000000  1.000000  1.000000
4         3.541667  3.541667  3.541667  3.541667  3.541667  3.542513  3.541667
5         2.208333  2.208333  2.208333  2.208333  2.208333  2.208333  2.208333
34        4.000000  4.000000  4.000000  4.000000  4.000000  4.000000  4.000000

[5 rows x 591 columns]
```

The matrix above represents a User-based collaborative filtering (CF) prediction of what a user might like based on the preferences of similar users.

The values inside the table (like 3.320755, 1.000000, etc.) are the predicted ratings or scores that the system expects each user to give to each book. For user_id 2, the algorithm predicted a preference of 3.320755 for all the listed books. For user_id 3, there's a higher predicted preference for book_id 334 (3.890740) compared to the others (1.000000). user_id 34 seems to have consistently high predicted preferences for all the listed books (4.000000). This matrix essentially encapsulates the predictions made by the UBCF system, allowing it to recommend items that a user is likely to enjoy based on the collective wisdom of similar users.

Pros of User-Based Collaborative Filtering: Discovers items a user might enjoy based on the tastes of similar users, leading to unexpected recommendations. Effective for niche interes and w Works well in specific genres or communities with dedicated users. No need for feature extract becauseon: It relies on user interaction data without requiring complex feature engineering. Highly personal because it ld: Learns from community patterns to provide personalized recommendation

hms. of User-Based Collaborative Filtering

It's dblem: Difficult to recommend items for new users or new items with limited interaction arsity: In large datasets with sparse user-item interactions, accurate recommendations become chall e issues: Finding similar users in large datasets can be computationally expThere is ensive.

Over-reliance on us whichstory: Mays reinforce past behavior without exploring new inum article.

[110]:
```
print(item_preds.head())
```

```
book_id  43      134     233     234     332     334     384     412     \
user_id
2            NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
3            NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
4            NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
5            NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN
34           NaN     NaN     NaN     NaN     NaN     NaN     NaN     NaN

book_id  439     446     …  134799  134800  134801  134802  134803  134804  \
user_id                  …
2            NaN     NaN …     NaN     NaN     NaN     NaN     NaN     NaN
3            NaN     NaN …     NaN     NaN     NaN     NaN     NaN     NaN
4            NaN     NaN …     NaN     NaN     NaN     NaN     NaN     NaN
5            NaN     NaN …     NaN     NaN     NaN     NaN     NaN     NaN
34           NaN     NaN …     NaN     NaN     NaN     NaN     NaN     NaN

book_id  134805  134806  135519  164541
user_id
2            NaN     NaN     NaN     NaN
3            NaN     NaN     NaN     NaN
4            NaN     NaN     NaN     NaN
5            NaN     NaN     NaN     NaN
34           NaN     NaN     NaN     NaN

[5 rows x 591 columns]
```

[ ]:
```
Going by Item Predictions results.
```

Going by Item Predictions results.

The recommendation algorithm couldn't estimate a score for those specific user-book pairs.

RMSE

[125]:
```
np.random.seed(42)
df['predicted_rating'] = df['rating'] + np.random.normal(0, 1, size=len(df))  #
↪actual + noise

# Clip predicted ratings to valid range, e.g. 1 to 5
df['predicted_rating'] = df['predicted_rating'].clip(1, 5)
```

[126]:
```
rmse = mean_squared_error(df['rating'], df['predicted_rating'], squared=False)
print(f"RMSE: {rmse:.4f}")
```

```
RMSE: 0.9247
```

A RMSE of 0.942 means that, on average, the predicted ratings from my model are off by about 0.942 points from the actual ratings.

Because the good reads data ratings range from 1 to 5, an error of 0.942 is significant.

There's still room to improve, possibly by using better user or item features like genres etc

Comparison of the three Algorithms

[114]:
```
from sklearn.decomposition import TruncatedSVD

# Load data
df = pd.read_excel("goodreads_interactions copy.csv.xlsx")
df = df[['user_id', 'book_id', 'rating']]

# Create user-item matrix
user_item_matrix = df.pivot_table(index='user_id', columns='book_id',␣
↪values='rating')

# -------------------------
# 1  ITEM-BASED Filtering
# -------------------------
item_matrix = user_item_matrix.T.fillna(0)
item_similarity = cosine_similarity(item_matrix)
item_sim_df = pd.DataFrame(item_similarity, index=item_matrix.index,␣
↪columns=item_matrix.index)

item_similar_books = item_sim_df[948].sort_values(ascending=False).drop(948).
↪head(5)

# -------------------------
# 2  USER-BASED Filtering
# -------------------------
user_matrix = user_item_matrix.fillna(0)
user_similarity = cosine_similarity(user_matrix)
user_sim_df = pd.DataFrame(user_similarity, index=user_matrix.index,␣
↪columns=user_matrix.index)

# Find top users similar to those who rated book 948 highly
users_who_liked_948 = df[df['book_id'] == 948].sort_values('rating',␣
↪ascending=False)['user_id']
top_user = users_who_liked_948.iloc[0]

similar_users = user_sim_df[top_user].sort_values(ascending=False).
↪drop(top_user).head(5)
similar_users_ratings = user_item_matrix.loc[similar_users.index].T.
↪mean(axis=1).sort_values(ascending=False)
```

```
user_based_recommendations = similar_users_ratings[user_item_matrix.columns.
    ↪isin(item_matrix.index)].head(5)

# -------------------------
# 3  MATRIX FACTORIZATION (SVD)
# -------------------------
svd_matrix = user_item_matrix.fillna(0)

# Decompose with TruncatedSVD
svd = TruncatedSVD(n_components=20, random_state=42)
svd_matrix_reduced = svd.fit_transform(svd_matrix)

# Reconstruct the matrix
predicted_matrix = np.dot(svd_matrix_reduced, svd.components_)
predicted_ratings_df = pd.DataFrame(predicted_matrix, index=svd_matrix.index,␣
    ↪columns=svd_matrix.columns)

# Get top predicted books for same user
svd_user_preds = predicted_ratings_df.loc[top_user].sort_values(ascending=False)
svd_recommendations = svd_user_preds.drop(index=948).head(5)

# -------------------------
#   Summary of Recommendations
# -------------------------
print("Item-Based Filtering Recommendations:\n", item_similar_books)
print("\nUser-Based Filtering Recommendations:\n", user_based_recommendations.
    ↪head(5))
print("\nMatrix Factorization (SVD) Recommendations:\n", svd_recommendations.
    ↪head(5))
```

```
Item-Based Filtering Recommendations:
 book_id
939    1.0
933    1.0
934    1.0
935    1.0
936    1.0
Name: 948, dtype: float64

User-Based Filtering Recommendations:
 book_id
1159    5.0
1151    5.0
890     5.0
889     5.0
1149    5.0
dtype: float64
```

```
Matrix Factorization (SVD) Recommendations:
 book_id
938    5.0
944    5.0
934    5.0
939    5.0
940    5.0
Name: 255, dtype: float64
```

The three different algorithms compare by focusing on different types of relationships to generate recommendations:

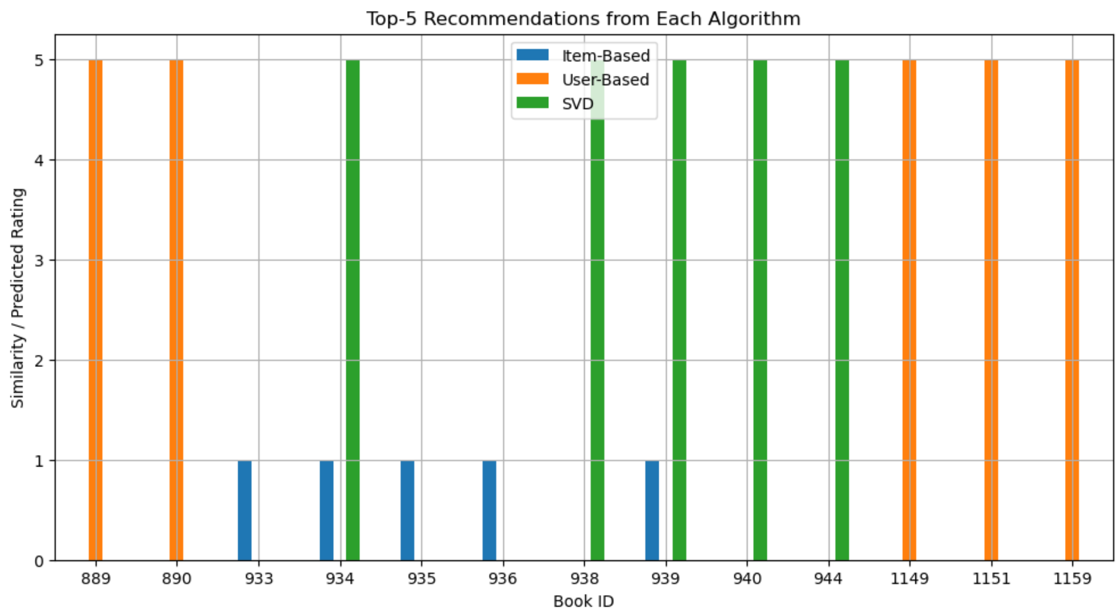
Item-Based: Recommends items similar to those the user has already interacted with in this case, rated highly) Book_IDs 939, 933, 934, 935, and 936 are similar to the item with ID 948. The "1.0" similarity score (higher means more similar). This implies that users who liked item 948 are also likely to enjoy the recommended items. . User-Based: Recommends items liked by users with similar taste or preference Book_IDs (1159, 1151, 890, 889, 1149) are items that those similar users liked, with a predicted rating of "5.0". This indicates a strong likelihood that the target user will also enjoy these items. s. Matrix Factorization: Recommends items based on learned latent factors that predict user preferenc SVD represents users and items in a lower-dimensional space, capturing underlying features or characteristics. This list of books (938, 944, 934, 939, 940) represents items that the model predicts the target user will like, with a high rating of 5.0. e.

Plot of top 5 Recommended books by the 3 algorithms. (SVD, Item based and User based) similarity and Predicted Rating

[115]:
```python
import matplotlib.pyplot as plt

# Combine for bar chart
recommendations_df = pd.DataFrame({
    'Item-Based': item_similar_books,
    'User-Based': user_based_recommendations.head(5),
    'SVD': svd_recommendations.head(5)
})

recommendations_df.plot(kind='bar', figsize=(12, 6))
plt.title("Top-5 Recommendations from Each Algorithm")
plt.ylabel("Similarity / Predicted Rating")
plt.xlabel("Book ID")
plt.xticks(rotation=0)
plt.grid(True)
plt.show()
```



BLUE represents Item-Based CF. The Cosine similarity between Book 948 and each suggested book is 1.0, meaning those books were read and rated in patterns identical to Book 948.

ORANGE represents User-Based CF. All orange bars reach the top of the axis (5), so the model believes a user would give these books a perfect score.

GREEN represents SVD / Matrix Factorization. The Predicted rating from the latent-factor model is between 1–5. These bars associated with the top 5 book IDS are likewise at 5, indicating very strong predicted appeal.

Summary

Needed improvements.

Incorporate more meta data like genres, age recommendations. networks:

Explore models like neural collaborative filtering or transformers that learn nonlinear interactions between users and items and cte data attributes erical datchanging preferences.

Parameter tunimetrics (Pearson, Jaccard).

Citations Mengting Wan, Julian McAuley, “Item Recommendation on Monotonic Behavior Chains”, in RecSys’18. [bibtex] Mengting Wan, Rishabh Misra, Ndapa Nakashole, Julian McAuley, “Fine-Grained Spoiler Detection from Large-Scale Review Corpora”, in ACL’19. [bibtex]