03_baseline_models

April 21, 2025

[1]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import sys
from pathlib import Path
import math # Import math

# Add project root to sys.path
project_root = Path.cwd().parent # Should be RECSYS_FINAL
src_path = project_root / "src"
sys.path.append(str(project_root)) # Add project root for imports like 'src.
 ↪config'

# Import project modules
from src import config
from src.data import preprocess # For time_based_split
from src.evaluation.evaluator import RecEvaluator # Import the evaluator class
from src.models.popularity import PopularityRecommender # Import the model

# Set display options
pd.set_option('display.max_columns', 100)
pd.set_option('display.max_rows', 100)
sns.set_style("whitegrid")
print("Setup complete. Modules imported.")
print(f"Project Root: {project_root}")
print(f"Processed Data Dir: {config.PROCESSED_DATA_DIR}")
```

```
Loading .env from: /Users/mohit/Desktop/everything/ATLAS/Semester
4/Pinnacle/recsys_final/.env
Database URI configured: Yes
Setup complete. Modules imported.
Project Root: /Users/mohit/Desktop/everything/ATLAS/Semester
4/Pinnacle/recsys_final
Processed Data Dir: /Users/mohit/Desktop/everything/ATLAS/Semester
4/Pinnacle/recsys_final/data/processed
```

[2]:

```
# Load the processed parquet files
try:
    interactions_df = pd.read_parquet(config.PROCESSED_DATA_DIR /␣
 ↪"interactions_final.parquet")
    users_df = pd.read_parquet(config.PROCESSED_DATA_DIR / "users_final.
 ↪parquet")
    items_df = pd.read_parquet(config.PROCESSED_DATA_DIR / "items_final.
 ↪parquet") # Contains presentation_id as column
    print("Processed data loaded successfully.")
    print(f"Interactions shape: {interactions_df.shape}")
    print(f"Users shape: {users_df.shape}")
    print(f"Items shape: {items_df.shape}")

    # Set presentation_id as index for items_df if needed later (evaluator uses␣
 ↪it)
    if 'presentation_id' in items_df.columns:
        items_df = items_df.set_index('presentation_id')
        print("Set 'presentation_id' as index for items_df.")

except FileNotFoundError as e:
    print(f"Error loading processed files: {e}")
    print("Please ensure the preprocessing pipeline (run_preprocessing.py) has␣
 ↪been run successfully.")
    # Stop execution or handle error
    raise e
except Exception as e:
    print(f"An unexpected error occurred during loading: {e}")
    raise e

# Display heads
print("\nInteractions Head:\n", interactions_df.head())
print("\nUsers Head:\n", users_df.head())
print("\nItems Head:\n", items_df.head())
```

```
Processed data loaded successfully.
Interactions shape: (28466, 7)
Users shape: (25364, 9)
Items shape: (22, 22)
Set 'presentation_id' as index for items_df.

Interactions Head:
    id_student presentation_id  total_clicks  interaction_days  \
0        6516       AAA_2014J          2791               159
1        8462       DDD_2013J           646                56
2        8462       DDD_2014J            10                 1
3       11391       AAA_2013J           934                40
4       23629       BBB_2013B           161                16
```

```
   first_interaction_date  last_interaction_date  implicit_feedback
0                     -23                    269           7.934513
1                      -6                    118           6.472346
2                      10                     10           2.397895
3                      -5                    253           6.840547
4                      -6                     87           5.087596

Users Head:
    id_student  num_of_prev_attempts  studied_credits  gender_mapped  \
0         6516                     0               60              0
1         8462                     1               60              0
2        11391                     0              240              0
3        23629                     2               60              1
4        23698                     0              120              1

   highest_education_mapped  imd_band_mapped  age_band_mapped  \
0                         3                9                2
1                         3                4                2
2                         3               10                2
3                         1                3                0
4                         2                6                0

   disability_mapped               region
0                  0             Scotland
1                  0        London Region
2                  0  East Anglian Region
3                  0  East Anglian Region
4                  0  East Anglian Region

Items Head:
                  module_presentation_length  vle_prop_dataplus  \
presentation_id
AAA_2013J                               268           0.018957
AAA_2014J                               269           0.019802
BBB_2013J                               268           0.000000
BBB_2014J                               262           0.000000
BBB_2013B                               240           0.000000

                 vle_prop_dualpane  vle_prop_externalquiz  vle_prop_folder  \
presentation_id
AAA_2013J                      0.0                    0.0              0.0
AAA_2014J                      0.0                    0.0              0.0
BBB_2013J                      0.0                    0.0              0.0
BBB_2014J                      0.0                    0.0              0.0
BBB_2013B                      0.0                    0.0              0.0

                 vle_prop_forumng  vle_prop_glossary  vle_prop_homepage  \
presentation_id
AAA_2013J                0.071090           0.009479           0.004739
AAA_2014J                0.029703           0.009901           0.004950
BBB_2013J                0.059190           0.003115           0.003115
BBB_2014J                0.014493           0.009662           0.004831
BBB_2013B                0.053968           0.003175           0.003175

                 vle_prop_htmlactivity  vle_prop_oucollaborate  \
presentation_id
AAA_2013J                          0.0                0.009479
AAA_2014J                          0.0                0.009901
BBB_2013J                          0.0                0.006231
BBB_2014J                          0.0                0.014493
BBB_2013B                          0.0                0.000000

                 vle_prop_oucontent  vle_prop_ouelluminate  vle_prop_ouwiki  \
presentation_id
AAA_2013J                  0.322275               0.000000              0.0
AAA_2014J                  0.336634               0.000000              0.0
BBB_2013J                  0.009346               0.000000              0.0
BBB_2014J                  0.338164               0.000000              0.0
BBB_2013B                  0.003175               0.003175              0.0

                 vle_prop_page  vle_prop_questionnaire  vle_prop_quiz  \
presentation_id
AAA_2013J                  0.0                0.000000       0.000000
AAA_2014J                  0.0                0.000000       0.000000
BBB_2013J                  0.0                0.000000       0.015576
BBB_2014J                  0.0                0.019324       0.019324
BBB_2013B                  0.0                0.000000       0.015873

                 vle_prop_repeatactivity  vle_prop_resource  \
presentation_id
AAA_2013J                            0.0           0.450237
AAA_2014J                            0.0           0.460396
BBB_2013J                            0.0           0.735202
BBB_2014J                            0.0           0.502415
BBB_2013B                            0.0           0.749206

                 vle_prop_sharedsubpage  vle_prop_subpage  vle_prop_url
presentation_id
AAA_2013J                      0.000000          0.028436      0.085308
AAA_2014J                      0.000000          0.029703      0.099010
BBB_2013J                      0.003115          0.118380      0.046729
BBB_2014J                      0.000000          0.048309      0.028986
BBB_2013B                      0.003175          0.117460      0.047619
```

[3]:

```
# Cell [3]: Time-Based Split (Using Threshold)

time_col = 'last_interaction_date'
user_col_in_df = 'id_student'      # Actual column name in interactions_df
item_col_in_df = 'presentation_id' # Actual column name in interactions_df

# --- Determine Threshold ---
print("--- Determining Time Threshold ---")
print(interactions_df[time_col].describe(percentiles=[.75, .8, .85, .9, .95]))
# Choose threshold based on percentiles (e.g., 80th percentile)
# ***** REPLACE 229 WITH YOUR CHOSEN VALUE *****
TIME_THRESHOLD = 250
print(f"Chosen Time Threshold: {TIME_THRESHOLD}")
print("--- End Threshold Determination ---")


# --- Perform Split ---
if time_col not in interactions_df.columns:
    raise ValueError(f"Time column '{time_col}' not found in interactions data.
 ↪")
if user_col_in_df not in interactions_df.columns:
    raise ValueError(f"User column '{user_col_in_df}' not found in interactions␣
 ↪data.")
if item_col_in_df not in interactions_df.columns:
    raise ValueError(f"Item column '{item_col_in_df}' not found in interactions␣
 ↪data.")
if not pd.api.types.is_numeric_dtype(interactions_df[time_col]):
     raise TypeError(f"Time column '{time_col}' must be numeric.")

train_df, test_df = preprocess.time_based_split(
    interactions_df=interactions_df,
    user_col=user_col_in_df,
    item_col=item_col_in_df,
    time_col=time_col,
    time_unit_threshold=TIME_THRESHOLD # <<< Use the threshold
    # split_ratio=None # Ensure split_ratio is not used
)

# --- Verify Split ---
print(f"\nTrain shape: {train_df.shape}")
print(f"Test shape: {test_df.shape}")
if not test_df.empty:
    print(f"Min time in Train: {train_df[time_col].min()}, Max time in Train:␣
 ↪{train_df[time_col].max()}")
    print(f"Min time in Test: {test_df[time_col].min()}, Max time in Test:␣
 ↪{test_df[time_col].max()}")
    # Check user/item overlap
    train_users_final = set(train_df[user_col_in_df].unique())
    test_users_final = set(test_df[user_col_in_df].unique())
    print(f"Users in Train: {len(train_users_final)}, Users in Test:␣
 ↪{len(test_users_final)}")
    print(f"Users ONLY in Test: {len(test_users_final - train_users_final)}") #␣
 ↪Should be 0 after filtering in split func

    train_items_final = set(train_df[item_col_in_df].unique())
    test_items_final = set(test_df[item_col_in_df].unique())
    print(f"Items in Train: {len(train_items_final)}, Items in Test:␣
 ↪{len(test_items_final)}")
    print(f"Items ONLY in Test: {len(test_items_final - train_items_final)}") #␣
 ↪Should be 0 after filtering in split func

else:
    print("Warning: Test DataFrame is empty!")
```

```
--- Determining Time Threshold ---
count    28466.000000
mean       180.275662
std         88.679680
min        -25.000000
50%        228.000000
75%        242.000000
80%        250.000000
85%        257.000000
90%        261.000000
95%        266.000000
max        269.000000
Name: last_interaction_date, dtype: float64
Chosen Time Threshold: 250
--- End Threshold Determination ---
Performing time-based split…
Original interactions shape: (28466, 7)
Splitting based on time threshold: last_interaction_date <= 250
 Initial train size: 22892, Initial test size: 5574
Filtered 4836 interactions from test set (users/items not in train).
Final Training set shape: (22892, 7)
Final Test set shape: (738, 7)
Users in Train: 20701, Users in Test: 731
Items in Train: 22, Items in Test: 13

Train shape: (22892, 7)
Test shape: (738, 7)
Min time in Train: -25, Max time in Train: 250
Min time in Test: 251, Max time in Test: 269
Users in Train: 20701, Users in Test: 731
Users ONLY in Test: 0
Items in Train: 22, Items in Test: 13
Items ONLY in Test: 0
```

[4]:

```
# Cell [4] - Train Popularity Model

# Initialize and train the Popularity model
# Ensure the item_col matches the column name in train_df and test_df
pop_model = PopularityRecommender(
    user_col='id_student',          # <<< Use the actual user column name
    item_col='presentation_id',     # <<< Use the actual item column name
    score_col='implicit_feedback'
)

# Fit the model using the training data
pop_model.fit(train_df)

# (Optional) Test prediction for a sample user/items
if not test_df.empty:
    sample_user = test_df['id_student'].iloc[0]
    sample_items_all = items_df.index.tolist() # Get all unique item IDs from␣
 ↪items_df index
    sample_items_subset = np.random.choice(sample_items_all, min(10,␣
 ↪len(sample_items_all)), replace=False).tolist() # Ensure not sampling more␣
 ↪than available
    print(f"\nTesting prediction for user {sample_user} on items:␣
 ↪{sample_items_subset}")
    scores = pop_model.predict(sample_user, sample_items_subset)
    print("Scores (Popularity):", scores)
else:
    print("\nSkipping prediction test as test_df is empty.")
```

```
Initialized PopularityRecommender
Fitting PopularityRecommender…
 Mapped 20701 users and 22 items.
Fit complete. Calculated popularity for 22 items.
Top 5 most popular items: ['FFF_2013B', 'CCC_2014B', 'FFF_2014B', 'BBB_2013B',
'BBB_2014J']

Testing prediction for user 29639 on items: ['GGG_2013J', 'EEE_2013J',
'DDD_2014B', 'GGG_2014J', 'FFF_2014B', 'DDD_2013J', 'EEE_2014J', 'BBB_2014B',
'AAA_2013J', 'EEE_2014B']
Scores (Popularity): [4234.305017098054, 3786.79398624287, 6704.788396041217,
3105.473475817197, 9406.690532184764, 7253.355677703635, 3896.5541039967225,
7208.319667495403, 1359.433747092062, 4083.0891111100705]
```

[5]:

```
# Cell [5] - Evaluate Popularity Model

# Initialize the evaluator
# Ensure items_df has presentation_id as index before passing
if test_df.empty:
     print("\nCannot evaluate model: Test data is empty.")
elif items_df.index.name != 'presentation_id':
     print("\nError: items_df must have 'presentation_id' set as index for␣
 ↪evaluator.")
else:
    evaluator = RecEvaluator(
        train_df=train_df,
        test_df=test_df,
        item_features_df=items_df, # Pass items_df with index set
        user_col='id_student',     # <<< Use the actual user column name
        item_col='presentation_id',# <<< Use the actual item column name
        k=config.TOP_K             # Use K from config
    )

    # Evaluate the popularity model
    # Using n_neg_samples can speed things up significantly for evaluation if␣
 ↪needed
    # Set n_neg_samples=100 for faster (approximate) evaluation, or None for␣
 ↪full evaluation
    print("\n--- Starting Evaluation of Popularity Model ---")
    pop_results = evaluator.evaluate_model(pop_model, n_neg_samples=100)

    print("\nPopularity Model Evaluation Results:")
    print(pop_results)
```

```
Evaluator initialized with 22 unique candidate items.
Stored 20701 training interactions for filtering.
Prepared test data for 731 users.

--- Starting Evaluation of Popularity Model ---

--- Evaluating Model: PopularityRecommender ---
Total test users: 731. Evaluating 731 users known by the model.

Evaluating users:   0%|          | 0/731 [00:00<?, ?it/s]


--- Evaluation Results (K=10) ---
Precision@10: 0.0621
Recall@10: 0.6156
NDCG@10: 0.2153
n_users_evaluated: 731.0000
n_users_skipped: 0.0000
```

```
------------------------------

Popularity Model Evaluation Results:
{'Precision@10': 0.06210670314637483, 'Recall@10': 0.615595075239398, 'NDCG@10':
0.2153109329329855, 'n_users_evaluated': 731}
```

[6]:

```
# Cell [6] - Train ItemCF Model

# Import the model
from src.models.item_cf import ItemCFRecommender

# Initialize and train the ItemCF model
itemcf_model = ItemCFRecommender(
    user_col='id_student',          # Use the actual user column name
    item_col='presentation_id',     # Use the actual item column name
    score_col='implicit_feedback'
)

# Fit the model using the training data
itemcf_model.fit(train_df)

# (Optional) Test prediction for a sample user/items
if not test_df.empty:
    # Use the same sample user as before or pick a new one
    sample_user_id = test_df['id_student'].iloc[0]
    # Ensure the user exists in the model's mapping
    if sample_user_id in itemcf_model.user_id_to_idx:
        # Get items the user interacted with in train and test for context
        user_train_interactions = train_df[train_df['id_student'] ==␣
↪sample_user_id]['presentation_id'].tolist()
        user_test_interactions = test_df[test_df['id_student'] ==␣
↪sample_user_id]['presentation_id'].tolist()
        print(f"\n--- ItemCF Prediction Test ---")
        print(f"Sample User ID: {sample_user_id}")
        print(f" User's Training Items: {user_train_interactions}")
        print(f" User's Test Items (Ground Truth): {user_test_interactions}")

        # Predict scores for the test items and a few others
        sample_items_all = items_df.index.tolist()
        items_to_predict = user_test_interactions + np.random.
↪choice(sample_items_all, 5, replace=False).tolist()
        items_to_predict = list(set(items_to_predict)) # Ensure unique items

        print(f" Predicting for Items: {items_to_predict}")
        scores = itemcf_model.predict(sample_user_id, items_to_predict)
        print(" Predicted Scores:", scores)
        print("--- End Prediction Test ---")
    else:
        print(f"Sample user {sample_user_id} not found in ItemCF model training␣
 ↪data.")

else:
    print("\nSkipping ItemCF prediction test as test_df is empty.")
```

```
Initialized ItemCFRecommender
Fitting ItemCFRecommender…
 Mapped 20701 users and 22 items.
Creating user-item interaction matrix…
 Created sparse matrix with shape: (20701, 22) and density: 0.0503
Calculating item-item cosine similarity…
 Calculated item similarity matrix shape: (22, 22)
Stored training interactions for prediction filtering.
Fit complete.

--- ItemCF Prediction Test ---
Sample User ID: 29639
 User's Training Items: ['EEE_2014B']
 User's Test Items (Ground Truth): ['CCC_2014J']
 Predicting for Items: ['CCC_2014J', 'FFF_2014J', 'EEE_2014B', 'AAA_2014J',
'DDD_2014B', 'FFF_2014B']
 Predicted Scores: [0.593949099458515, 0.020948052775677548, 0.0, 0.0, 0.0, 0.0]
--- End Prediction Test ---
```

[7]:

```
# Cell [7] - Evaluate ItemCF Model

# Evaluate the ItemCF model using the same evaluator instance
if 'evaluator' in locals() and evaluator is not None: # Check if evaluator␣
 ↪exists
    print("\n--- Starting Evaluation of ItemCF Model ---")
    itemcf_results = evaluator.evaluate_model(itemcf_model, n_neg_samples=100)␣
 ↪# Use negative sampling

    print("\nItemCF Model Evaluation Results:")
    print(itemcf_results)

elif test_df.empty:
    print("\nCannot evaluate model: Test data is empty.")
else:
     print("\nError: Evaluator not initialized. Please run Cell [5]␣
 ↪successfully first.")
```

```
--- Starting Evaluation of ItemCF Model ---
--- Evaluating Model: ItemCFRecommender ---
Total test users: 731. Evaluating 731 users known by the model.

Evaluating users:   0%|          | 0/731 [00:00<?, ?it/s]


--- Evaluation Results (K=10) ---
Precision@10: 0.0988
Recall@10: 0.9781
NDCG@10: 0.6153
n_users_evaluated: 731.0000
n_users_skipped: 0.0000
------------------------------

ItemCF Model Evaluation Results:
{'Precision@10': 0.09876880984952119, 'Recall@10': 0.9781121751025992,
'NDCG@10': 0.6152957703109161, 'n_users_evaluated': 731}
```

[8]:

```python
# Cell [10] - Train ItemCF Model

# Import the model
from src.models.item_cf import ItemCFRecommender

print("\n--- Training ItemCF Model ---")

# Initialize and train the ItemCF model
itemcf_model = ItemCFRecommender(
    user_col='id_student',          # Use the actual user column name from␣
 ↪interactions_df
    item_col='presentation_id',     # Use the actual item column name from␣
 ↪interactions_df
    score_col='implicit_feedback'
)

# Fit the model using the training data
# This might take a moment as it calculates the similarity matrix
itemcf_model.fit(train_df)

# (Optional) Test prediction for a sample user/items
if not test_df.empty:
    # Use the same sample user as before or pick a new one from the test set
    sample_user_id = test_df['id_student'].iloc[0] # Example: first user in␣
 ↪test set

    # Ensure the user exists in the model's mapping
    if sample_user_id in itemcf_model.user_id_to_idx:
        # Get items the user interacted with in train and test for context
        user_train_interactions = train_df[train_df['id_student'] ==␣
 ↪sample_user_id]['presentation_id'].tolist()
        user_test_interactions = test_df[test_df['id_student'] ==␣
 ↪sample_user_id]['presentation_id'].tolist()
        print(f"\n--- ItemCF Prediction Test ---")
        print(f"Sample User ID: {sample_user_id}")
        print(f" User's Training Items: {user_train_interactions}")
        print(f" User's Test Items (Ground Truth): {user_test_interactions}")

        # Predict scores for the test items and a few others
        sample_items_all = items_df.index.tolist()
        items_to_predict = user_test_interactions + np.random.
 ↪choice(sample_items_all, 5, replace=False).tolist()
        items_to_predict = list(set(items_to_predict)) # Ensure unique items

        print(f" Predicting for Items: {items_to_predict}")
        scores = itemcf_model.predict(sample_user_id, items_to_predict)
        print(" Predicted Scores:", scores)
        # Display scores alongside item IDs for better readability
        scored_preds = sorted(list(zip(items_to_predict, scores)), key=lambda x:
 ↪ x[1], reverse=True)
        print(" Predicted Scores (Sorted):", scored_preds)
        print("--- End Prediction Test ---")
    else:
        print(f"\nSample user {sample_user_id} not found in ItemCF model␣
 ↪training data (this shouldn't happen if test set was filtered correctly).")
else:
    print("\nSkipping ItemCF prediction test as test_df is empty.")

print("\n--- Finished Training ItemCF Model ---")
```

```
--- Training ItemCF Model ---
Initialized ItemCFRecommender
Fitting ItemCFRecommender…
 Mapped 20701 users and 22 items.
Creating user-item interaction matrix…
 Created sparse matrix with shape: (20701, 22) and density: 0.0503
Calculating item-item cosine similarity…
 Calculated item similarity matrix shape: (22, 22)
Stored training interactions for prediction filtering.
Fit complete.

--- ItemCF Prediction Test ---
Sample User ID: 29639
 User's Training Items: ['EEE_2014B']
 User's Test Items (Ground Truth): ['CCC_2014J']
 Predicting for Items: ['CCC_2014J', 'FFF_2013J', 'EEE_2013J', 'DDD_2013B',
'GGG_2014B', 'FFF_2014B']
 Predicted Scores: [0.593949099458515, 0.008426447318283632, 0.0645435753807183,
0.016319825324461853, 0.0, 0.0]
 Predicted Scores (Sorted): [('CCC_2014J', 0.593949099458515), ('EEE_2013J',
0.0645435753807183), ('DDD_2013B', 0.016319825324461853), ('FFF_2013J',
0.008426447318283632), ('GGG_2014B', 0.0), ('FFF_2014B', 0.0)]
--- End Prediction Test ---

--- Finished Training ItemCF Model ---
```

[9]:

```python
# Cell [11] - Evaluate ItemCF Model

# Evaluate the ItemCF model using the same evaluator instance
if 'evaluator' in locals() and evaluator is not None: # Check if evaluator␣
 ↪exists
    print("\n--- Starting Evaluation of ItemCF Model ---")
    itemcf_results = evaluator.evaluate_model(itemcf_model, n_neg_samples=100)␣
 ↪# Use negative sampling

    print("\nItemCF Model Evaluation Results:")
    print(itemcf_results)

elif test_df.empty:
    print("\nCannot evaluate ItemCF model: Test data is empty.")
else:
     print("\nError: Evaluator not initialized. Please run the cell that␣
 ↪initializes 'evaluator' successfully first.")
```

```
--- Starting Evaluation of ItemCF Model ---

--- Evaluating Model: ItemCFRecommender ---
Total test users: 731. Evaluating 731 users known by the model.

Evaluating users:   0%|          | 0/731 [00:00<?, ?it/s]


--- Evaluation Results (K=10) ---
Precision@10: 0.0988
Recall@10: 0.9781
NDCG@10: 0.6153
n_users_evaluated: 731.0000
n_users_skipped: 0.0000
------------------------------

ItemCF Model Evaluation Results:
{'Precision@10': 0.09876880984952119, 'Recall@10': 0.9781121751025992,
'NDCG@10': 0.6152957703109161, 'n_users_evaluated': 731}
```

[10]:

```
# Cell [12] - Train ALS Model

# Import the model
from src.models.matrix_factorization import ImplicitALSWrapper

print("\n--- Training Implicit ALS Model ---")

# Initialize and train the ALS model
# Adjust hyperparameters as needed (these are examples)
als_model = ImplicitALSWrapper(
    user_col='id_student',
    item_col='presentation_id',
    score_col='implicit_feedback',
    factors=50,            # Latent factors
    regularization=0.05,   # Regularization
    iterations=25,         # Iterations
    random_state=config.RANDOM_SEED
)

# Fit the model using the training data
# This will take longer than Popularity or ItemCF
als_model.fit(train_df)

# (Optional) Test prediction for a sample user/items
if not test_df.empty:
    sample_user_id = test_df['id_student'].iloc[0]
    if sample_user_id in als_model.user_id_to_idx:
        user_train_interactions = train_df[train_df['id_student'] ==␣
 ↪sample_user_id]['presentation_id'].tolist()
        user_test_interactions = test_df[test_df['id_student'] ==␣
 ↪sample_user_id]['presentation_id'].tolist()
        print(f"\n--- ALS Prediction Test ---")
        print(f"Sample User ID: {sample_user_id}")
        print(f" User's Training Items: {user_train_interactions}")
        print(f" User's Test Items (Ground Truth): {user_test_interactions}")

        sample_items_all = items_df.index.tolist()
        items_to_predict = user_test_interactions + np.random.
 ↪choice(sample_items_all, 5, replace=False).tolist()
        items_to_predict = list(set(items_to_predict))

        print(f" Predicting for Items: {items_to_predict}")
        scores = als_model.predict(sample_user_id, items_to_predict)
        scored_preds = sorted(list(zip(items_to_predict, scores)), key=lambda x:
 ↪ x[1], reverse=True)
        print(" Predicted Scores (Sorted):", scored_preds)
        print("--- End Prediction Test ---")
    else:
        print(f"\nSample user {sample_user_id} not found in ALS model training␣
 ↪data.")
else:
    print("\nSkipping ALS prediction test as test_df is empty.")

print("\n--- Finished Training Implicit ALS Model ---")
```

```
--- Training Implicit ALS Model ---
Initialized ImplicitALSWrapper
Fitting ImplicitALSWrapper…
 Mapped 20701 users and 22 items.
Creating user-item interaction matrix for Implicit ALS…
 Created sparse matrix (Users x Items) shape: (20701, 22) density: 0.0503
Initializing implicit.als.AlternatingLeastSquares…
Fitting model on User x Item matrix shape: (20701, 22)…
```

```
/opt/anaconda3/lib/python3.12/site-packages/implicit/cpu/als.py:95:
RuntimeWarning: OpenBLAS is configured to use 16 threads. It is highly
recommended to disable its internal threadpool by setting the environment
variable 'OPENBLAS_NUM_THREADS=1' or by calling
'threadpoolctl.threadpool_limits(1, "blas")'. Having OpenBLAS use a threadpool
can lead to severe performance issues here.
  check_blas_config()
```

```
  0%|          | 0/25 [00:00<?, ?it/s]
```

```
Model fitting complete.

--- ALS Prediction Test ---
Sample User ID: 29639
 User's Training Items: ['EEE_2014B']
 User's Test Items (Ground Truth): ['CCC_2014J']
 Predicting for Items: ['CCC_2014J', 'DDD_2013B', 'AAA_2013J', 'EEE_2014J',
'CCC_2014B', 'FFF_2014B']
 Predicted Scores (Sorted): [('CCC_2014B', 0.0017379806376993656), ('EEE_2014J',
0.0004142490215599537), ('CCC_2014J', 0.00020702210895251483), ('DDD_2013B',
0.00016349671932402998), ('AAA_2013J', -6.1340538195509e-06), ('FFF_2014B',
-9.699559450382367e-05)]
--- End Prediction Test ---

--- Finished Training Implicit ALS Model ---
```

[11]:

```
# Cell [13] - Evaluate ALS Model

# Evaluate the ALS model using the same evaluator instance
if 'evaluator' in locals() and evaluator is not None:
    print("\n--- Starting Evaluation of Implicit ALS Model ---")
    als_results = evaluator.evaluate_model(als_model, n_neg_samples=100) # Use␣
 ↪negative sampling

    print("\nImplicit ALS Model Evaluation Results:")
    print(als_results)

elif test_df.empty:
    print("\nCannot evaluate ALS model: Test data is empty.")
else:
     print("\nError: Evaluator not initialized. Please run the cell that␣
 ↪initializes 'evaluator' successfully first.")
```

```
--- Starting Evaluation of Implicit ALS Model ---

--- Evaluating Model: ImplicitALSWrapper ---
Total test users: 731. Evaluating 731 users known by the model.

Evaluating users:   0%|          | 0/731 [00:00<?, ?it/s]


--- Evaluation Results (K=10) ---
Precision@10: 0.0685
Recall@10: 0.6778
NDCG@10: 0.3844
n_users_evaluated: 731.0000
n_users_skipped: 0.0000
------------------------------

Implicit ALS Model Evaluation Results:
{'Precision@10': 0.06853625170998631, 'Recall@10': 0.6778385772913816,
'NDCG@10': 0.3844081787750316, 'n_users_evaluated': 731}
```