Author

- Name: Su Qiu Lin
- Number: 72405483

Content Introduction

This part is about Collaborative Filtering(itemcf/usercf) and Matrix Factorization(svd), based on two datasets: lastfm and ml32m.

- To run and reproduction
  1. Install the requirements.txt
  2. Rename datasets and modify the path of the dataset in the code(Optional)
     Due to the difference of operating systems, the path may be different.
     - def load_lastfm: **path="./datas/hetrec2011-lastfm-2k"** or **path="../datas/hetrec2011-lastfm-2k"**
     - def load_ml32m: **path="./datas/ml-32m"** or **path="../datas/ml-32m"**
  3. Run lastfm.py and ml32m.py file independently.
- This ipynb file is just a snapshot of the two .py files' computational results.

Project Structure

collaborative/

- lastfm.py
- ml32m.py
- readme.ipynb
- requirements.txt - list of dependencies, using "pip install -r requirements.txt" to install all dependencies

datas/

- hetrec2011-lastfm-2k/
  - user_artists.dat
  - artists.dat
- ml-32m/
  - ratings.csv
  - movies.csv

Summary

Method	Advantages	Disadvantages
ItemCF	• Good performance for stable item preferences • Strong interpretability of recommendations • No need to recalculate when new users join • Well-suited for items with rich features • Pre-computed similarities can be stored	• Poor performance with sparse item data • Performance degrades with large item catalogs • Cold-start problem for new items • Cannot capture latent factors • Limited diversity in recommendations
UserCF	• Works well with sparse user data • Can discover users' emerging interests • Sensitive to changes in user preferences • Intuitive algorithm concept • Good for social recommendations	• Poor scalability with large user bases • Sensitive to popular items • Cold-start problem for new users • Requires frequent recalculation • Accuracy decreases when user interests are diverse
MF (Matrix Factorization)	• Strong ability to handle sparse data • Can discover latent feature factors • Good scalability for large datasets • High prediction accuracy • Effective with high-dimensional features	• Poor interpretability of recommendations • Cold-start problem for new users/items • Requires hyperparameter tuning • High computational cost for model training • Difficult to update in real-time

Experiment Snapshot

1. lastfm.py - ItemCF/UserCF/SVD

In [6]:
```python
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from collections import defaultdict
import random
from tqdm import tqdm
from scipy.sparse.linalg import svds

# Load the dataset
def load_lastfm(path="../datas/hetrec2011-lastfm-2k"):
    # Load user-artist interactions
    user_artists = pd.read_csv(f"{path}/user_artists.dat", sep='\t')

    # Load artists data
    artists = pd.read_csv(f"{path}/artists.dat", sep='\t')

    return user_artists, artists

# Preprocess data
def preprocess_data(user_artists):
    # Create user-item matrix
```

```
    user_item_matrix_df = user_artists.pivot(index='userID', columns='art
    return user_item_matrix_df

# User-based collaborative filtering
# 根据与user_id的相似用户，推荐n个item（artist）给它
def user_based_cf(user_item_matrix_df, user_id, n_users=10, n_recommendat
    # Calculate user similarity
    user_similarity = cosine_similarity(user_item_matrix_df)
    user_similarity_df = pd.DataFrame(user_similarity, index=user_item_ma

    # Find similar users
    similar_users = user_similarity_df[user_id].sort_values(ascending=Fal

    # Get recommendations
    recommendations = defaultdict(float)

    # 对于当前没听过的艺术家，以相似用户喜欢的艺术家频次为准，
    # recommendations[item] = 累加（某user和其相似用户的相似度 * weight）
    for similar_user in similar_users:
        similarity_between_user = user_similarity_df.loc[user_id, similar

        for item in user_item_matrix_df.columns:
            if user_item_matrix_df.loc[user_id, item] == 0 and user_item_
                recommendations[item] += similarity_between_user * user_i

    # Sort recommendations
    recommendations = sorted(recommendations.items(), key=lambda x: x[1],
    return recommendations

# Item-based collaborative filtering
def item_based_cf(user_item_matrix_df, user_id, n_items=10, n_recommendat
    # 1. Calculate item similarity between items
    item_similarity = cosine_similarity(user_item_matrix_df.T)
    item_similarity_df = pd.DataFrame(item_similarity, index=user_item_ma

    # 2. Get items the user has interacted with and weights > 0
    user_related_items = user_item_matrix_df.loc[user_id]
    related_items_id = user_related_items[user_related_items > 0].index.t

    # Get recommendations
    recommendations = defaultdict(float)

    for item_id in related_items_id:
        item_weight = user_item_matrix_df.loc[user_id, item_id]

        # 2.1 Find n top items similar to current item, according to item
        similar_items = item_similarity_df[item_id].sort_values(ascending

        for similar_item, similarity_between_item in similar_items.items(
            # 2.2 only find the similar item is not in related_items_id,
            if similar_item in related_items_id:
                continue
            # the below similar_item's weight must be 0
            recommendations[similar_item] += similarity_between_item * it

    # Sort recommendations
    recommendations = sorted(recommendations.items(), key=lambda x: x[1],
    return recommendations

# SVD-based collaborative filtering
```

```python
def svd_based_cf(user_item_matrix_df, user_id, n_factors=50, n_recommenda
    # Convert to numpy array for SVD
    user_item_array = user_item_matrix_df.values

    # Get user index
    user_idx = user_item_matrix_df.index.get_loc(user_id)

    # Ensure n_factors is not too large
    n_factors = min(n_factors, min(user_item_array.shape) - 1)

    # Perform SVD
    u, sigma, vt = svds(user_item_array, k=n_factors)

    # Convert to diagonal matrix
    sigma_diag = np.diag(sigma)

    # Reconstruct the matrix using the factors
    reconstructed_matrix = np.dot(np.dot(u, sigma_diag), vt)

    # Get the reconstructed ratings for the user
    user_ratings = reconstructed_matrix[user_idx]

    # Get actual ratings for the user
    actual_ratings = user_item_array[user_idx]

    # Create a mask of items the user hasn't interacted with
    unrated_items = actual_ratings == 0

    # Get indices of items the user hasn't interacted with, ordered by pr
    candidate_items = np.argsort(-user_ratings * unrated_items)[:n_recomm

    # Convert back to original item IDs and predicted scores
    recommendations = [(user_item_matrix_df.columns[item_idx], user_ratin
                        for item_idx in candidate_items if unrated_items[it

    return recommendations

# Evaluate recommendations
def evaluate(user_item_matrix, test_ratio=0.2, n_users=10, n_items=10, n_
    # Create a copy of the matrix to avoid modifying the original
    matrix = user_item_matrix.copy()

    # Metrics storage
    metrics = {
        'user_hr': [], 'user_ndcg': [], 'user_mrr': [],
        'item_hr': [], 'item_ndcg': [], 'item_mrr': [],
        'svd_hr': [], 'svd_ndcg': [], 'svd_mrr': []
    }
    # if not sample, cause 3 hours to execute evaluate but get similar re
    sampled_users = random.sample(list(matrix.index), 50)
    print(f"Evaluating recommendations for {len(sampled_users)} users..."

    # For each user, hide some interactions as test data
    for user_id in tqdm(sampled_users, desc="Evaluating", ncols=80):
        # Get items this user has interacted with
        user_items = matrix.columns[matrix.loc[user_id] > 0].tolist()

        # Skip users with too few interactions
        if len(user_items) <= 2:
            continue
```

```
        # Randomly select items for testing
        n_test = max(1, int(len(user_items) * test_ratio))
        test_items = random.sample(user_items, n_test)

        # Create training matrix by setting test items to zero
        train_matrix = matrix.copy()
        for item in test_items:
            train_matrix.loc[user_id, item] = 0

        # Get recommendations from all methods
        user_recs = user_based_cf(train_matrix, user_id, n_users, n_recom
        item_recs = item_based_cf(train_matrix, user_id, n_items, n_recom
        svd_recs = svd_based_cf(train_matrix, user_id, n_factors, n_recom

        # Extract just the item IDs
        user_rec_items = [item_id for item_id, _ in user_recs]
        item_rec_items = [item_id for item_id, _ in item_recs]
        svd_rec_items = [item_id for item_id, _ in svd_recs]

        # Calculate metrics for user-based CF
        metrics['user_hr'].append(hit_ratio(user_rec_items, test_items))
        metrics['user_ndcg'].append(ndcg(user_rec_items, test_items))
        metrics['user_mrr'].append(mrr(user_rec_items, test_items))

        # Calculate metrics for item-based CF
        metrics['item_hr'].append(hit_ratio(item_rec_items, test_items))
        metrics['item_ndcg'].append(ndcg(item_rec_items, test_items))
        metrics['item_mrr'].append(mrr(item_rec_items, test_items))

        # Calculate metrics for SVD-based CF
        metrics['svd_hr'].append(hit_ratio(svd_rec_items, test_items))
        metrics['svd_ndcg'].append(ndcg(svd_rec_items, test_items))
        metrics['svd_mrr'].append(mrr(svd_rec_items, test_items))

    # Calculate average metrics
    avg_metrics = {k: np.mean(v) for k, v in metrics.items() if v}

    return avg_metrics

# Hit Ratio@K
def hit_ratio(recommended_items, test_items):
    hits = len(set(recommended_items) & set(test_items))
    return hits / len(test_items) if test_items else 0

# NDCG@K
def ndcg(recommended_items, test_items):
    dcg = 0
    idcg = 0

    # Calculate DCG
    for i, item in enumerate(recommended_items):
        if item in test_items:
            # Using binary relevance (1 if hit, 0 if miss)
            dcg += 1 / np.log2(i + 2)  # i+2 because i starts from 0

    # Calculate IDCG (ideal DCG - items are perfectly ranked)
    for i in range(min(len(test_items), len(recommended_items))):
        idcg += 1 / np.log2(i + 2)
```

```
    return dcg / idcg if idcg > 0 else 0

# MRR@K
def mrr(recommended_items, test_items):
    for i, item in enumerate(recommended_items):
        if item in test_items:
            return 1 / (i + 1)  # i+1 because i starts from 0
    return 0

def load_and_preprocess_data():
    user_artists, artists = load_lastfm()
    user_item_matrix_df = preprocess_data(user_artists)
    return user_artists, artists, user_item_matrix_df

def print_recommendations(user_id, recommendations, artists, method_name)
    print(f"\n{method_name} recommendations for user {user_id}:")
    for item_id, score in recommendations:
        artist_name = artists[artists['id'] == item_id]['name'].values[0]
        print(f"Artist ID: {item_id}, Score: {score:.2f}, Name: {artist_n

def generate_recommendations(user_item_matrix_df, user_id, artists):
    # Get recommendations using different methods
    user_recommendations = user_based_cf(user_item_matrix_df, user_id)
    print_recommendations(user_id, user_recommendations, artists, "User-b

    item_recommendations = item_based_cf(user_item_matrix_df, user_id)
    print_recommendations(user_id, item_recommendations, artists, "Item-b

    svd_recommendations = svd_based_cf(user_item_matrix_df, user_id)
    print_recommendations(user_id, svd_recommendations, artists, "SVD-bas

def evaluate_models(user_item_matrix_df):
    print("\nEvaluating recommendation methods...")
    metrics = evaluate(user_item_matrix_df)
    print(f"\nEvaluation results:")
    print(f"User-based CF - HR@K: {metrics['user_hr']:.4f}, NDCG@K: {metr
    print(f"Item-based CF - HR@K: {metrics['item_hr']:.4f}, NDCG@K: {metr
    print(f"SVD-based CF - HR@K: {metrics['svd_hr']:.4f}, NDCG@K: {metric

def main():
    # Load and preprocess data
    user_artists, artists, user_item_matrix_df = load_and_preprocess_data

    # Example: Get recommendations for a specific user
    user_id = user_item_matrix_df.index[0]  # First user in the dataset

    # Generate and print recommendations
    generate_recommendations(user_item_matrix_df, user_id, artists)

    # Evaluate the models
    evaluate_models(user_item_matrix_df)

if __name__ == "__main__":
    main()
```

```
User-based recommendations for user 2:
Artist ID: 511, Score: 11033.60, Name: U2
Artist ID: 159, Score: 9444.82, Name: The Cure
Artist ID: 1001, Score: 8645.83, Name: Pet Shop Boys
Artist ID: 2562, Score: 6262.49, Name: Arcadia
Artist ID: 1014, Score: 5252.81, Name: Erasure
Artist ID: 993, Score: 5032.88, Name: Simple Minds
Artist ID: 187, Score: 4859.54, Name: a-ha
Artist ID: 4313, Score: 4470.64, Name: Nephew
Artist ID: 227, Score: 3800.56, Name: The Beatles
Artist ID: 6776, Score: 3569.32, Name: Book of Love

Item-based recommendations for user 2:
Artist ID: 2556, Score: 14102.14, Name: The Power Station
Artist ID: 8995, Score: 13904.49, Name: Andy Taylor
Artist ID: 1076, Score: 13796.74, Name: Wham!
Artist ID: 2562, Score: 13181.35, Name: Arcadia
Artist ID: 13161, Score: 12606.11, Name: Private
Artist ID: 4313, Score: 12600.67, Name: Nephew
Artist ID: 6350, Score: 12455.23, Name: TV-2
Artist ID: 996, Score: 11801.53, Name: Mike & The Mechanics
Artist ID: 4042, Score: 6047.56, Name: Damn Arms
Artist ID: 4023, Score: 6047.56, Name: DJ Risk One

SVD-based recommendations for user 2:
Artist ID: 3501, Score: 2290.23, Name: Mylène Farmer
Artist ID: 1098, Score: 1818.59, Name: Björk
Artist ID: 159, Score: 1556.17, Name: The Cure
Artist ID: 265, Score: 1553.20, Name: Céline Dion
Artist ID: 1001, Score: 1109.75, Name: Pet Shop Boys
Artist ID: 475, Score: 1072.39, Name: Eminem
Artist ID: 1505, Score: 1050.33, Name: Natalie Imbruglia
Artist ID: 7759, Score: 1035.20, Name: Bushido
Artist ID: 228, Score: 886.41, Name: Kings of Leon
Artist ID: 173, Score: 866.37, Name: Placebo

Evaluating recommendation methods...
Evaluating recommendations for 50 users...
Evaluating: 100%|██████████████████████████████| 50/50 [05:20<00:00,  6.4
2s/it]
Evaluation results:
User-based CF - HR@K: 0.1213, NDCG@K: 0.1421, MRR@K: 0.3611
Item-based CF - HR@K: 0.0180, NDCG@K: 0.0237, MRR@K: 0.0833
SVD-based CF - HR@K: 0.0949, NDCG@K: 0.1048, MRR@K: 0.2782
```

2. ml32m.py - UserCF/SVD

not suitable for itemCF because the movie items are too many

In [2]:

```python
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from collections import defaultdict
import random
from tqdm import tqdm
from scipy.sparse import csr_matrix
from scipy.sparse.linalg import svds
```

```
"""
author: suqiulin-72405483
- Perform collaborative recommendation algorithm(usercf/svd) on this data
    - itemCF won't be implement because there are 292757 movies(item) whi
"""
# Load the ML-32M dataset
def load_ml32m(path="../datas/ml-32m"):
    print("Loading ML-32M dataset...")
    # Load ratings data in chunks due to large size
    ratings = pd.read_csv(f"{path}/ratings.csv", chunksize=1000000)
    ratings_df = pd.concat(ratings)
    # Optional: load movie data if available
    try:
        movies = pd.read_csv(f"{path}/movies.csv")
    except:
        movies = None
        print("Movies data not found. Will continue without movie titles.

    return ratings_df, movies

# Preprocess data
def preprocess_data(ratings_df, sample_size=None):
    print("Preprocessing data...")
    if sample_size:
        # Take a sample if full dataset is too large
        user_counts = ratings_df['userId'].value_counts()
        users_to_keep = user_counts[user_counts >= 5].index[:sample_size]
        ratings_df = ratings_df[ratings_df['userId'].isin(users_to_keep)]

    # Create sparse user-item matrix
    users = ratings_df['userId'].unique()
    items = ratings_df['movieId'].unique()

    user_mapper = {user: i for i, user in enumerate(users)}
    item_mapper = {item: i for i, item in enumerate(items)}

    user_indices = [user_mapper[user] for user in ratings_df['userId']]
    item_indices = [item_mapper[item] for item in ratings_df['movieId']]

    # Create sparse matrix
    matrix = csr_matrix((ratings_df['rating'], (user_indices, item_indice
                         shape=(len(users), len(items)))

    return matrix, users, items, user_mapper, item_mapper

# User-based collaborative filtering
def user_based_cf(matrix, user_idx, n_users=10, n_recommendations=10):
    # Get user row
    user_row = matrix[user_idx].toarray().flatten()

    # Calculate similarities
    similarities = []
    for i in range(matrix.shape[0]):
        if i != user_idx:
            other_row = matrix[i].toarray().flatten()
            # Compute similarity only if users have common items
            if np.sum(user_row > 0) > 0 and np.sum(other_row > 0) > 0:
                sim = cosine_similarity(user_row.reshape(1, -1), other_ro
                similarities.append((i, sim))
```

```
    # Sort by similarity
    similarities.sort(key=lambda x: x[1], reverse=True)
    similar_users = similarities[:n_users]

    # Get recommendations
    recommendations = defaultdict(float)
    for similar_user, similarity in similar_users:
        similar_user_row = matrix[similar_user].toarray().flatten()
        for item_idx in range(len(similar_user_row)):
            if user_row[item_idx] == 0 and similar_user_row[item_idx] > 0
                recommendations[item_idx] += similarity * similar_user_ro

    # Sort recommendations
    recommendations = sorted(recommendations.items(), key=lambda x: x[1],
    return recommendations

# Item-based collaborative filtering
def item_based_cf(matrix, user_idx, n_items=10, n_recommendations=10):
    # Get user's rated items
    user_row = matrix[user_idx].toarray().flatten()
    user_rated_items = np.where(user_row > 0)[0]

    # Get recommendations
    recommendations = defaultdict(float)
    for item_idx in user_rated_items:
        item_col = matrix.T[item_idx].toarray().flatten()

        # Find similar items
        for other_item_idx in range(matrix.shape[1]):
            if other_item_idx != item_idx and user_row[other_item_idx] ==
                other_item_col = matrix.T[other_item_idx].toarray().flatt
                # Compute similarity only if items have common users
                if np.sum(item_col > 0) > 0 and np.sum(other_item_col > 0
                    sim = cosine_similarity(item_col.reshape(1, -1), othe
                    recommendations[other_item_idx] += sim * user_row[ite

    # Sort recommendations
    recommendations = sorted(recommendations.items(), key=lambda x: x[1],
    return recommendations

# SVD-based collaborative filtering
def svd_based_cf(matrix, user_idx, n_factors=50, n_recommendations=10):
    # Perform SVD
    U, sigma, Vt = svds(matrix, k=n_factors)

    # Reconstruct the matrix
    sigma_diag = np.diag(sigma)
    predicted_ratings = np.dot(np.dot(U, sigma_diag), Vt)

    # Get the predicted ratings for user
    user_pred_ratings = predicted_ratings[user_idx]

    # Get actual ratings
    user_row = matrix[user_idx].toarray().flatten()

    # Create mask for unrated items
    unrated_items = np.where(user_row == 0)[0]

    # Get recommendations
```

```
    recommendations = [(item_idx, user_pred_ratings[item_idx])
                       for item_idx in unrated_items]

    # Sort by predicted rating
    recommendations.sort(key=lambda x: x[1], reverse=True)
    return recommendations[:n_recommendations]

# Evaluation metrics
def hit_ratio(recommended_items, test_items):
    hits = len(set(recommended_items) & set(test_items))
    return hits / len(test_items) if test_items else 0

def ndcg(recommended_items, test_items):
    dcg = 0
    idcg = 0

    for i, item in enumerate(recommended_items):
        if item in test_items:
            dcg += 1 / np.log2(i + 2)

    for i in range(min(len(test_items), len(recommended_items))):
        idcg += 1 / np.log2(i + 2)

    return dcg / idcg if idcg > 0 else 0

def mrr(recommended_items, test_items):
    for i, item in enumerate(recommended_items):
        if item in test_items:
            return 1 / (i + 1)
    return 0

# Evaluate recommendations
def evaluate(matrix, users, test_ratio=0.2, n_users=10, n_items=10, n_fac
    metrics = {
        'user_hr': [], 'user_ndcg': [], 'user_mrr': [],
        'item_hr': [], 'item_ndcg': [], 'item_mrr': [],
        'svd_hr': [], 'svd_ndcg': [], 'svd_mrr': []
    }

    # Sample users for evaluation
    sampled_users = random.sample(range(len(users)), min(50, len(users)))
    print(f"Evaluating recommendations for {len(sampled_users)} users..."

    for user_idx in tqdm(sampled_users, desc="Evaluating", ncols=80):
        # Get items user has rated
        user_row = matrix[user_idx].toarray().flatten()
        user_items = np.where(user_row > 0)[0]

        # Skip users with too few interactions
        if len(user_items) <= 2:
            continue

        # Select test items
        n_test = max(1, int(len(user_items) * test_ratio))
        test_items = random.sample(list(user_items), n_test)

        # Create training matrix
        train_matrix = matrix.copy()
        for item_idx in test_items:
            train_matrix[user_idx, item_idx] = 0
```

```
        # Get recommendations
        user_recs = user_based_cf(train_matrix, user_idx, n_users, n_reco
        # item_recs = item_based_cf(train_matrix, user_idx, n_items, n_re
        svd_recs = svd_based_cf(train_matrix, user_idx, n_factors, n_reco

        # Extract just the item indices
        user_rec_items = [item_idx for item_idx, _ in user_recs]
        svd_rec_items = [item_idx for item_idx, _ in svd_recs]

        # Calculate metrics
        metrics['user_hr'].append(hit_ratio(user_rec_items, test_items))
        metrics['user_ndcg'].append(ndcg(user_rec_items, test_items))
        metrics['user_mrr'].append(mrr(user_rec_items, test_items))

        metrics['svd_hr'].append(hit_ratio(svd_rec_items, test_items))
        metrics['svd_ndcg'].append(ndcg(svd_rec_items, test_items))
        metrics['svd_mrr'].append(mrr(svd_rec_items, test_items))

    # Calculate average metrics
    avg_metrics = {k: np.mean(v) for k, v in metrics.items() if v}
    return avg_metrics

def print_recommendations(user_id, recommendations, original_items, movie
    print(f"\n{method_name} recommendations for user {user_id}:")
    for item_idx, score in recommendations:
        item_id = original_items[item_idx]
        if movies is not None and item_id in movies['movieId'].values:
            movie_name = movies[movies['movieId'] == item_id]['title'].va
            print(f"Movie ID: {item_id}, Score: {score:.2f}, Title: {movi
        else:
            print(f"Movie ID: {item_id}, Score: {score:.2f}")

def main():
    # Load and preprocess data
    ratings_df, movies = load_ml32m()

    # Use a sample of the data if it's too large
    # (may generate 16966441536 which will crush the program)
    # Adjust based on your hardware capabilities
    sample_size = 10000
    print(f"Using a sample of {sample_size} users for analysis...")

    matrix, users, items, user_mapper, item_mapper = preprocess_data(rati

    # Example: Get recommendations for a specific user
    user_idx = 0  # First user in the sample
    original_user_id = users[user_idx]

    # Generate recommendations
    print(f"\nGenerating recommendations for user {original_user_id}...")

    user_recs = user_based_cf(matrix, user_idx)
    print_recommendations(original_user_id, user_recs, items, movies, "Us

    svd_recs = svd_based_cf(matrix, user_idx)
    print_recommendations(original_user_id, svd_recs, items, movies, "SVD

    # Evaluate models
    print("\nEvaluating recommendation methods...")
```

```
    metrics = evaluate(matrix, users)

    print(f"\nEvaluation results:")
    print(f"User-based CF - HR@K: {metrics['user_hr']:.4f}, NDCG@K: {metr
    print(f"SVD-based CF - HR@K: {metrics['svd_hr']:.4f}, NDCG@K: {metric

if __name__ == "__main__":
    main()
```

```
Loading ML-32M dataset...
Using a sample of 10000 users for analysis...
Preprocessing data...

Generating recommendations for user 10...

User-based recommendations for user 10:
Movie ID: 3147, Score: 19.81, Title: Green Mile, The (1999)
Movie ID: 54286, Score: 19.78, Title: Bourne Ultimatum, The (2007)
Movie ID: 8961, Score: 19.76, Title: Incredibles, The (2004)
Movie ID: 1676, Score: 19.75, Title: Starship Troopers (1997)
Movie ID: 2858, Score: 19.70, Title: American Beauty (1999)
Movie ID: 4011, Score: 19.51, Title: Snatch (2000)
Movie ID: 59784, Score: 19.50, Title: Kung Fu Panda (2008)
Movie ID: 69844, Score: 19.23, Title: Harry Potter and the Half-Blood Prin
ce (2009)
Movie ID: 4886, Score: 19.18, Title: Monsters, Inc. (2001)
Movie ID: 64614, Score: 18.96, Title: Gran Torino (2008)

SVD-based recommendations for user 10:
Movie ID: 69844, Score: 3.15, Title: Harry Potter and the Half-Blood Princ
e (2009)
Movie ID: 81834, Score: 2.92, Title: Harry Potter and the Deathly Hallows:
Part 1 (2010)
Movie ID: 88125, Score: 2.79, Title: Harry Potter and the Deathly Hallows:
Part 2 (2011)
Movie ID: 54286, Score: 2.67, Title: Bourne Ultimatum, The (2007)
Movie ID: 2858, Score: 2.54, Title: American Beauty (1999)
Movie ID: 3147, Score: 2.45, Title: Green Mile, The (1999)
Movie ID: 1136, Score: 2.33, Title: Monty Python and the Holy Grail (1975)
Movie ID: 3535, Score: 2.24, Title: American Psycho (2000)
Movie ID: 60074, Score: 2.23, Title: Hancock (2008)
Movie ID: 30793, Score: 2.20, Title: Charlie and the Chocolate Factory (20
05)

Evaluating recommendation methods...
Evaluating recommendations for 50 users...
Evaluating: 100%|██████████| 50/50 [08:56<00:00, 10.7
3s/it]
Evaluation results:
User-based CF - HR@K: 0.0434, NDCG@K: 0.7641, MRR@K: 0.9667
SVD-based CF - HR@K: 0.0525, NDCG@K: 0.8940, MRR@K: 0.9567
```