collaborative_filtering

November 17, 2024

[110]:
```python
import pandas as pd
# Evaluate user-user and item-item collaborative filtering using the MovieLens␣
↪100K dataset,
# which contains 100,000 ratings from 943 users on 1,682 movies.
# Load the dataset 'u_data.csv' file from working directory after download.
columns = ['user_id', 'item_id', 'rating', 'timestamp']
data = pd.read_csv('u.data', sep='\t', names=columns)

# The csv contains four columns of data:
# --user_id - the unique identification number assigned to each movie goer/
↪movie reviewer.
# --item_id - the unique identification number assigned to each movie that has␣
↪been reviewed or rated.
# --rating - the rating assigned by a movie reviewer to a movie on a scale of 1␣
↪to 5.
# --timestamp - Unix timestamp (date and time) for when the movie was rated.
# The full data set is 10,000 rows with each row holding a movie reviewer's ID␣
↪number,
# the ID number of the movie reviewed, the rating of the movie between 1 and 5,␣
↪the timestamp of the review.

# Check the dataset structure
print(data.head())
print(data.tail())
```

```
   user_id  item_id  rating  timestamp
0      196      242       3  881250949
1      186      302       3  891717742
2       22      377       1  878887116
3      244       51       2  880606923
4      166      346       1  886397596
       user_id  item_id  rating  timestamp
99995      880      476       3  880175444
99996      716      204       5  879795543
99997      276     1090       1  874795795
99998       13      225       2  882399156
99999       12      203       3  879959583
```

```
[112]: import numpy as np
# Create Ratings Matrix
# Create a user-item matrix dataframe from csv data.
# The csv data is converted into a basic ratings matrix or "user_item_matrix"␣
↪using the pivot function
# and missing values are filled with NaN. The pivot function constructs a new␣
↪dataframe with movie
# reviewer IDs (user_ids) along the index or y-axis as rows, movie IDs␣
↪(item_ids) along the upper
# x-axis as columns, and movie ratings (rating) as the actual values in the␣
↪matrix.
user_item_matrix = data.pivot(index='user_id', columns='item_id',␣
↪values='rating').fillna(0)

# Convert non-numeric dataframe into numeric numpy array (or matrix) for later␣
↪work with machine learning algorithms.
# Each column is an item rating vector  that can be used in item-item␣
↪collaborative filtering
#    in which users similar to a target user are sought in order to make␣
↪predictions about the target user.
# Each row is a user rating vector that can be used in user-user collaborative␣
↪filtering
#    in which items similar to a target item are sought in order to make␣
↪predictions about the target item.
user_item_matrix = user_item_matrix.values

# Check shape. The matrix is now of m x n dimensions with m rows of users or␣
↪number of movie reviewers (num_users)
# and n colummns of rated movies.
print("User-Item Matrix shape:", user_item_matrix.shape)
```

```
User-Item Matrix shape: (943, 1682)
```

```
[114]: # Ratings Matrix Factorization
# Import modules and specific algorithm for Singular Value Decomposition (SVD)␣
↪matrix decomposition or factorization
# The original user_item_matrix with m users and n movie items is factorized␣
↪into three smaller matrices:
#                 user_item_matrix   User Matrix (m x k) * Latent Feature␣
↪Matrix (k x k) * Item Matrix (k x n)
# --User Matrix of dimensions m x k with m movie reviewer rows and k latent␣
↪feature columns.
# --Latent Feature Matrix of dimensions k x k as a diagonal matrix of the top␣
↪most important k latent features.
# --Item Matrix of dimensions k x n with k latent feature rows and n rated␣
↪movie columns.
```

```
# Latent features are hidden patterns or abstract concepts in the data that␣
↪explain the relationships
# between users and items in a recommendation system. They are not explicitly␣
↪present in the dataset
# but are derived mathematically during techniques like SVD.

from sklearn.decomposition import TruncatedSVD

# Create an model instance of the TruncatedSVD class.
# Decompose or factorize with SVD.
# Set the number of derived latent features or components (n_components) to 20.
# This is the number of latent factors or features that will be retained when␣
↪performing dimensionality reduction.
# The arbitrary value of "20" is selected based on experimentation or␣
↪heuristics to balance model complexity and performance.
# Higher n_components values capture more variance but may lead to overfitting␣
↪or computational inefficiency,
# while lower values may oversimplify the data.
svd = TruncatedSVD(n_components=20)

# Fit the model instance to the ratings matrix, "user_item_matrix"
user_latent_matrix = svd.fit_transform(user_item_matrix)
# The resulting latent_matrix represents a version of the original␣
↪user_item_matrix that has been transformed by the SVD model
# into a reduced-dimensional format with dimensions (num_users, n_components).
# This new matrix contains the user representations in the latent feature␣
↪space, a compressed space that captures
# important patterns in the data.
# Each row corresponds to a user or movie rewiewer, and each column represents␣
↪a latent feature (one of the 20 n_components).

# Show shape of reduced-dimension, latent feature space version of original␣
↪rating matrix
print("Latent Matrix shape:", user_latent_matrix.shape)
```

```
Latent Matrix shape: (943, 20)
```

[116]:
```
# Use reduced-dimension, latent feature space version of original rating matrix␣
↪for user-user collaborative filtering
# Function to recommend items based on user-user similarity
def recommend_for_user(user_id, user_latent_matrix, user_similarity, top_n=20):␣
↪# top_n similar users
    """
    Finds users similar to a given target user based on cosine similarity.

    Parameters:
```

```
    - user_id: The ID of the user to find similarities for.
    - user_latent_matrix: Matrix of latent features for all users.
    - top_n: Number of similar users to return.

    Returns:
    - List of IDs of the top_n most similar users.
    """
    # Find the most similar users
    similar_users = np.argsort(user_similarity[user_id])[-top_n-1:-1]  #␣
↪Exclude self (last user in sorted array)

    # Get the ratings of the most similar users (example)
    user_ratings = np.zeros(user_latent_matrix.shape[1])  # Placeholder for␣
↪ratings across items
    for similar_user in similar_users:
        user_ratings += user_latent_matrix[similar_user]  # Aggregate ratings␣
↪from similar users

    # Rank the items and recommend the top ones
    recommended_items = np.argsort(user_ratings)[-top_n:]  # Get the top N items
    return recommended_items
```

[119]:

```
# Determine impact of centered and uncentered cosine similarity on user-user␣
↪movie recommendations
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# Get movie names and IDs
columns = ['movie id', 'movie title', 'release date', 'video release date',
              'IMDb URL', 'unknown', 'Action', 'Adventure', 'Animation',
              'Childrens', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy',
              'Film-Noir', 'Horror', 'Musical', 'Mystery', 'Romance', 'Sci-Fi',
              'Thriller', 'War', 'Western']
movie_data = pd.read_csv('u.item', sep='|', names=columns,␣
↪encoding='ISO-8859-1')

# List of user IDs to test
numbers = [34, 205, 390, 174, 910]

for index, i in enumerate(numbers):
    user_id = i
    # Compute cosine similarity between users using latent_matrix from SVD
    user_similarity = cosine_similarity(latent_matrix)

    # Get the top recommended item IDs for the user
    recommended_items = recommend_for_user(user_id, user_latent_matrix,␣
↪user_similarity, top_n=5)
```

```
    # Get the movie names corresponding to the recommended movie IDs
    recommended_movie_names = movie_data[movie_data['movie id'].
↪isin(recommended_items)]['movie title'].values

    print(f"Recommended items for User {user_id} using cosine similarity:")
    for movie in recommended_movie_names:
        print(f"- {movie}")

    # Compute centered cosine similarity between users using the centered␣
↪latent_matrix
    # Center the latent matrix by subtracting the mean rating of each user
    mean_user_rating = np.mean(user_latent_matrix, axis=1)  # Calculate mean␣
↪rating for each user
    centered_user_latent_matrix = user_latent_matrix - mean_user_rating[:, np.
↪newaxis]  # Subtract mean from each user's ratings
    centered_user_similarity = cosine_similarity(centered_user_latent_matrix)

    # Get the top recommended item IDs for the user
    centered_recommended_items = recommend_for_user(user_id,␣
↪centered_user_latent_matrix, centered_user_similarity, top_n=5)

    # Get the movie names corresponding to the recommended movie IDs (centered␣
↪similarity)
    centered_recommended_movie_names = movie_data[movie_data['movie id'].
↪isin(centered_recommended_items)]['movie title'].values

    print(f"Recommended items for User {user_id} using centered cosine␣
↪similarity:")
    for movie in centered_recommended_movie_names:
        print(f"- {movie}")
    print(f"\n")
```

```
Recommended items for User 34 using cosine similarity:
- GoldenEye (1995)
- Four Rooms (1995)
- Get Shorty (1995)
- Seven (Se7en) (1995)
Recommended items for User 34 using centered cosine similarity:
- GoldenEye (1995)
- Four Rooms (1995)
- Get Shorty (1995)
- Seven (Se7en) (1995)


Recommended items for User 205 using cosine similarity:
- GoldenEye (1995)
```

```
- Four Rooms (1995)
- Get Shorty (1995)
- Postino, Il (1994)
Recommended items for User 205 using centered cosine similarity:
- GoldenEye (1995)
- Four Rooms (1995)
- Get Shorty (1995)
- Postino, Il (1994)


Recommended items for User 390 using cosine similarity:
- Toy Story (1995)
- GoldenEye (1995)
- Richard III (1995)
- Antonia's Line (1995)
Recommended items for User 390 using centered cosine similarity:
- Toy Story (1995)
- GoldenEye (1995)
- Seven (Se7en) (1995)
- Antonia's Line (1995)


Recommended items for User 174 using cosine similarity:
- Toy Story (1995)
- Usual Suspects, The (1995)
- Postino, Il (1994)
- Mr. Holland's Opus (1995)
Recommended items for User 174 using centered cosine similarity:
- Toy Story (1995)
- Usual Suspects, The (1995)
- Postino, Il (1994)
- Mr. Holland's Opus (1995)


Recommended items for User 910 using cosine similarity:
- Toy Story (1995)
- Four Rooms (1995)
- Twelve Monkeys (1995)
- White Balloon, The (1995)
Recommended items for User 910 using centered cosine similarity:
- Toy Story (1995)
- Four Rooms (1995)
- Twelve Monkeys (1995)
- From Dusk Till Dawn (1996)
```

[121]:
```
# Use reduced-dimension, latent feature space version of original rating matrix␣
↪for movie-movie collaborative filtering
# Fit the model instance to the ratings matrix, "user_item_matrix.T" to create␣
↪movie_latent_matrix
# This time the original ratings matrix is transposed to emphasize movies␣
↪rather than users
movie_latent_matrix = svd.fit_transform(user_item_matrix.T)
```

[123]:
```
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# Function to find similar movies
def find_similar_movies(target_movie_id, latent_matrix, top_n=5):
    """
    Finds movies similar to a given target movie based on cosine similarity.

    Parameters:
    - target_movie_id: The ID of the movie to find similarities for.
    - latent_matrix: Matrix of latent features for all items (movies).
    - top_n: Number of similar movies to return.

    Returns:
    - List of IDs of the top_n most similar movies.
    """
    # Compute cosine similarity between the target movie and all other movies
    movie_similarity = cosine_similarity(movie_latent_matrix)
    similarity_scores = movie_similarity[target_movie_id]

    # Get indices of top_n most similar movies (excluding the target movie␣
↪itself)
    similar_movie_ids = np.argsort(similarity_scores)[-top_n-1:-1][::-1]  #␣
↪Sort and exclude the target movie

    return similar_movie_ids
```

[125]:
```
# List of movie IDs to test
numbers = [34, 205, 390, 174, 910]

for index, i in enumerate(numbers):
    # movie_latent_matrix defined from the SVD step above
    target_movie_id = i  # Example target movie
    target_movie_name = movie_data[movie_data['movie id'].
↪isin([target_movie_id])]['movie title'].values
    top_similar_movies = find_similar_movies(target_movie_id,␣
↪movie_latent_matrix, top_n=5)

    # Get the movie names corresponding to the recommended similar movie IDs
```

```
    top_similar_movies = movie_data[movie_data['movie id'].
↪isin(top_similar_movies)]['movie title'].values

    print(f"Movies similar to Movie {target_movie_name}:␣
↪{top_similar_movies}\n")
```

```
Movies similar to Movie ['Doom Generation, The (1995)']: ['Big Night (1996)'
'Love Bug, The (1969)' 'Three Musketeers, The (1993)'
 'Addiction, The (1995)' 'Grumpier Old Men (1995)']

Movies similar to Movie ['Patton (1970)']: ['Fargo (1996)' 'Godfather, The
(1972)' 'Alien (1979)'
 'Ruling Class, The (1972)' 'Cutthroat Island (1995)']

Movies similar to Movie ['Fear of a Black Hat (1993)']: ['Apollo 13 (1995)'
'Three Colors: White (1994)' 'Jeffrey (1995)'
 'Speechless (1994)' 'Rising Sun (1993)']

Movies similar to Movie ['Raiders of the Lost Ark (1981)']: ['Sleepless in
Seattle (1993)' '12 Angry Men (1957)'
 'Young Frankenstein (1974)' 'James and the Giant Peach (1996)'
 'Crossfire (1947)']

Movies similar to Movie ['Nil By Mouth (1997)']: ['Big Lebowski, The (1998)'
'Half Baked (1998)' 'Dangerous Beauty (1998)'
 'Twilight (1998)' 'Mercury Rising (1998)']
```