Latihan dan Evaluasi Ujian Akhir

Implementasi Komprehensif dari Sistem Rekomendasi Collaborative Filtering models

untuk memenuhi Ujian Akhir Semester
Mata Kuliah Komputasi Lanjut dan Big Data

Dosen Pengampu:

Dr. Risman Adnan

Disusun Oleh:

Eka Dwi Agustina Ginting 2206103390

Program Magister Matematika

Fakultas Matematika dan Ilmu Pengetahuan Alam

Universitas Indonesia

2022

Daftar Isi

Kata Pengantar

Segala puji bagi Allah yang telah memberikan kemudahan dan kesempatan kepada saya untuk menyusun buku ini. Kami mengucapkan terima kasih kepada-Nya atas bimbingan dan pertolongan-Nya selama proses penyusunan buku ini.

Sebagai pengantar untuk buku kumpulan tugas komputasi lanjut dan big data ini, Saya ingin menyampaikan terima kasih kepada semua pihak yang telah membantu dalam proses penyusunan buku ini.

Pertama, dengan segala hormat, kami mengucapkan terima kasih yang sebesar-besarnya kepada Dosen yang telah memberikan bimbingan dan arahan yang sangat bermanfaat dalam mata kuliah komputasi lanjut dan big data ini. saya sangat menghargai dedikasi dan komitmen Dosen dalam menyampaikan materi yang berkualitas serta membantu saya dalam memahami materi pembelajaran komputasi lanjut dan big data yang benar dan baik.

saya juga sangat berterima kasih atas waktu yang Dosen khususkan untuk memberikan arahan dan bimbingan kepada kami secara individual.Saya yakin bahwa dengan bimbingan dan arahan yang telah diberikan oleh Dosen,saya akan mampu Menyusun makalah tugas komputasi lanjut dan big data yang berkualitas dan dapat memberikan manfaat bagi masyarakat. Terima kasih atas perhatian dan bantuan yang telah diberikan oleh Dosen. Saya berharap dapat terus memperoleh dukungan dan bimbingan dari Dosen di masa yang akan datang.

Kedua, saya mengucapkan terima kasih kepada penulis yang telah memberikan sumbangsihnya dalam bentuk makalah yang terkumpul dalam buku ini.Saya menghargai dedikasi dan komitmen mereka dalam menyusun makalah yang berkualitas.

Dengan adanya buku ini, kami harap dapat memberikan manfaat bagi para penulis dan pembaca yang akan mempelajarinya.Saya berharap buku ini dapat memberikan wawasan baru bagi pembaca dan menjadi sumber inspirasi bagi para penulis untuk terus berkarya dan mengembangkan ilmu pengetahuan di bidangnya masing-masing. Terima kasih atas perhatian dan waktu yang Anda berikan untuk membaca buku ini. Saya berharap dapat memberikan pengalaman membaca yang menyenangkan bagi Anda.

Salam,
Penyusun Buku

BAB 1
Pendahuluan

Teknologi komputer telah mengalami perkembangan yang sangat pesat selama beberapa dekade terakhir. Inovasi baru terus muncul dari waktu ke waktu, yang membantu meningkatkan kemampuan komputer dan membuat teknologi lebih mudah diakses oleh masyarakat luas. Salah satu contoh perkembangan teknologi komputer adalah pertumbuhan AI (Artificial Intelligence) atau kecerdasan buatan. AI memungkinkan komputer untuk belajar, beradaptasi, dan memecahkan masalah secara otomatis, yang membantu meningkatkan kemampuan komputer dalam berbagai bidang, seperti kesehatan, transportasi, keuangan, dan media.

Teknologi komputer juga telah membantu meningkatkan konektivitas dan kolaborasi antar individu dan organisasi. Platform seperti email, aplikasi pesan instan, dan layanan cloud storage memungkinkan orang untuk bekerja secara terintegrasi dari jarak yang jauh. Di masa depan, diperkirakan teknologi komputer akan terus berkembang dan memberikan inovasi baru yang membantu memudahkan kehidupan manusia. Namun, dengan semakin tingginya tingkat ketergantungan terhadap teknologi, penting untuk mempertimbangkan dampaknya terhadap privasi dan keamanan individu serta masyarakat.

Sistem rekomendasi dengan AI adalah sistem yang menggunakan algoritma pembelajaran mesin untuk memprediksi preferensi atau kecenderungan pengguna berdasarkan data yang dikumpulkan tentang perilaku pengguna sebelumnya. AI dapat membantu meningkatkan akurasi rekomendasi dengan menganalisis data yang lebih banyak dan lebih detail, sehingga dapat memberikan rekomendasi yang lebih sesuai dengan preferensi pengguna. Sistem rekomendasi dengan AI dapat digunakan dalam berbagai bidang, termasuk e-commerce, layanan streaming, dan media sosial. Di situs e-commerce, sistem rekomendasi dengan AI dapat membantu meningkatkan penjualan dengan memberikan rekomendasi produk yang sesuai dengan preferensi pengguna. Di layanan streaming, sistem rekomendasi dengan AI dapat membantu meningkatkan pengalaman pengguna dengan menyarankan film atau acara TV yang sesuai dengan kecenderungan pengguna. Di media sosial, sistem rekomendasi dengan AI dapat membantu meningkatkan kepuasan pengguna dengan menyarankan konten yang sesuai dengan preferensi pengguna.

Meskipun sistem rekomendasi dengan AI dapat membantu meningkatkan pengalaman pengguna dan menghasilkan lebih banyak penjualan, penting untuk diingat bahwa sistem ini hanya dapat memberikan rekomendasi berdasarkan data yang tersedia. Oleh karena itu, penting untuk memastikan bahwa data yang digunakan untuk membangun sistem rekomendasi dengan AI merupakan data yang akurat dan representative. Pada makalah ini akan dibahas lebih lajut mengenai metode system rekomendasi Reproduksi Collaborative Filtering models.

BAB 2

Tinjauan Pustaka

2.1.Definisi Sistem Rekomendasi

Sistem pendukung keputusan merupakan sebuah sistem berbasis komputer yang interaktif dalam membantu individu pengambil keputusan dengan memanfaatkan data dan model untuk penyelesaian masalah yang tidak terstuktur dan salah satu bentuk sistem pendukung keputusan adalah adanya sistem rekomendasi [1].

Sistem rekomendasi adalah sistem atau aplikasi dari hasil rekomendasi yang tepat yang digunakan untuk menyarankan item atau konten yang dianggap sesuai dengan preferensi atau keinginann pengguna berdasarkan data yang dikumpulkan tentang perilaku pengguna sebelumnya. Oleh karena itu sistem rekomendasi akan menawarkan kemungkinan dari penyaringan informasi personal sehingga hanyak informasi yang sesuai dengan kebutuhan dan preferensi pengguna yang akan ditampilkan di sistem dengan menggunakan sebuah teknik atau model rekomendasi [2].

2.2.Metode Sistem Rekomendasi

Dalam system rekomendasi, setiap metode yang digunakan disesuaikan dengan permasalahan dalam menghasilkan sebuah informasi yang sesuai. Metode atau pendekatan yang dipilih pada sistem rekomendasi bergantung pada permasalahan yang akan diselesaikan, teknik rekomendasi yang berbeda-beda digunakan untuk aplikasi yang berbeda, dasar dari suatu tujuan dan objektif dari sebuah aplikasi [2].

Ada beberapa metode yang dapat digunakan dalam sistem rekomendasi, di antaranya adalah: [3]

1. **Collaborative filtering**

   Collaborative filtering adalah salah satu metode yang sering digunakan dalam sistem rekomendasi untuk menentukan preferensi pengguna dan memberikan rekomendasi kepada pengguna berdasarkan preferensi yang sama dengan pengguna lain. Collaborative filtering menggunakan data interaksi pengguna dengan item sebagai dasar untuk menentukan preferensi pengguna [3],[4],[5].

Collaborative filtering dapat dibagi menjadi dua jenis, yaitu user-based collaborative filtering dan item-based collaborative filtering. Pada User-based collaborative filtering adalah metode yang mencari pengguna yang memiliki preferensi yang sama dengan pengguna yang sedang dianalisis, dan memberikan rekomendasi berdasarkan item yang disukai oleh pengguna tersebut [6]. Untuk menggunakan user-based collaborative filtering, pertama-tama harus tersedia data interaksi pengguna dengan item. Kemudian, sistem akan mencari pengguna yang memiliki preferensi yang sama dengan pengguna yang sedang dianalisis, dan menghitung similarity score antara kedua pengguna tersebut. Setelah itu, sistem akan menyaring item yang belum pernah dikonsumsi oleh pengguna yang sedang dianalisis, tetapi telah dikonsumsi oleh pengguna lain yang memiliki preferensi yang sama, dan memberikan rekomendasi berdasarkan item tersebut [7].

Sedangkan pada Item-based collaborative filtering adalah metode yang mencari item yang memiliki preferensi yang sama dengan item yang telah dikonsumsi oleh pengguna yang sedang dianalisis, dan memberikan rekomendasi berdasarkan item tersebut. Untuk menggunakan item-based collaborative filtering, pertama-tama harus tersedia data interaksi pengguna dengan item. Kemudian, sistem akan mencari item yang memiliki preferensi yang sama dengan item yang telah dikonsumsi oleh pengguna yang sedang dianalisis, dan menghitung similarity score antara kedua item tersebut. Setelah itu, sistem akan menyaring pengguna yang belum pernah mengonsumsi item yang sedang dianalisis, tetapi telah mengonsumsi item lain yang memiliki preferensi yang sama, dan memberikan rekomendasi berdasarkan pengguna tersebut [4].

Untuk menghitung similarity score antara pengguna atau item dalam collaborative filtering, dapat digunakan beberapa metode, seperti Pearson correlation, cosine similarity, atau metode lainnya. Setelah similarity score terhitung, sistem dapat memberikan rekomendasi berdasarkan pengguna atau item yang memiliki similarity score tertinggi [6].

Kelebihan collaborative filtering adalah dapat menghasilkan rekomendasi yang sesuai dengan preferensi pengguna, dan dapat menangani cold start problem dengan memanfaatkan data dari pengguna lain yang memiliki preferensi yang sama [7]. Namun, collaborative filtering juga memiliki kekurangan, seperti dapat menghasilkan rekomendasi yang bias terhadap item populer, membutuhkan cukup banyak data tentang interaksi pengguna dengan item untuk menghasilkan rekomendasi yang akurat, dan dapat mengalami scaling problem jika ada banyak pengguna dan item, Cold-start problem, karena

pendekatan collaborative filtering melakukan prediksi berdasarkan rating yang diberikan user pada item, maka menjadi suatu masalah ketika suatu item baru masuk ke dalam sistem dan belum di-rating sama sekali oleh user. Akibatnya item tersebut tidak akan pernah direkomendasikan kepada user. Sparsity, untuk ukuran data yang besar, banyak item yang baru sedikit di-rating oleh user, akibatnya item tersebut memiliki nilai prediksi yang relatif tidak akurat dan m enghasilkan rekomendasi yang buruk [8].

2. Content-based filtering

Content-based filtering adalah salah satu metode yang digunakan dalam sistem rekomendasi untuk memberikan rekomendasi kepada pengguna berdasarkan karakteristik item yang telah dikonsumsi oleh pengguna tersebut. Dalam content-based filtering, sistem akan mengumpulkan informasi tentang karakteristik item yang telah dikonsumsi oleh pengguna, kemudian mencari item yang memiliki karakteristik yang sama dengan item yang telah dikonsumsi oleh pengguna tersebut, dan memberikan rekomendasi berdasarkan item tersebut.

Untuk mengimplementasikan sistem rekomendasi content-based filtering, pertama-tama harus tersedia data tentang karakteristik item yang akan direkomendasikan. Kemudian, sistem akan mengumpulkan informasi tentang karakteristik item yang telah dikonsumsi oleh pengguna, kemudian mencari item yang memiliki karakteristik yang sama dengan item yang telah dikonsumsi oleh pengguna tersebut, dan memberikan rekomendasi berdasarkan item tersebut [3].

Untuk menghitung similarity score antara item dalam content-based filtering, dapat digunakan beberapa metode, seperti cosine similarity, Jaccard coefficient, atau metode lainnya. Setelah similarity score terhitung, sistem dapat memberikan rekomendasi berdasarkan item yang memiliki similarity score tertinggi [3].

Content-based filtering memiliki beberapa kelebihan, seperti tidak memerlukan data interaksi pengguna dengan item yang lain, dan dapat memberikan rekomendasi yang lebih tepat sesuai dengan preferensi pengguna. Namun, content-based filtering juga memiliki kelemahan, seperti tidak dapat mengambil keuntungan dari preferensi pengguna yang lain, dan tergantung pada kualitas data yang tersedia tentang karakteristik item [3].

3. Hybrid filtering

Hybrid filtering adalah salah satu metode yang digunakan dalam sistem rekomendasi untuk mengombinasikan beberapa metode rekomendasi, seperti collaborative filtering dan content-based filtering, dan memberikan rekomendasi berdasarkan kombinasi metode tersebut. Hybrid filtering dapat memberikan rekomendasi yang lebih tepat dibandingkan dengan hanya menggunakan satu metode saja, karena dapat mengambil keuntungan dari kelebihan masing-masing metode [3].

Untuk mengimplementasikan sistem rekomendasi hybrid filtering, pertama-tama harus tersedia data interaksi pengguna dengan item, serta data tentang karakteristik item yang akan direkomendasikan. Kemudian, sistem akan menggunakan collaborative filtering untuk mencari pengguna atau item yang memiliki preferensi yang sama dengan pengguna atau item yang sedang dianalisis, dan menghitung similarity score antara kedua pengguna atau item tersebut. Setelah itu, sistem akan menggunakan content-based filtering untuk mencari item yang memiliki karakteristik yang sama dengan item yang telah dikonsumsi oleh pengguna, dan menghitung similarity score antara kedua item tersebut. Setelah itu, sistem akan memberikan rekomendasi berdasarkan kombinasi similarity score yang diperoleh dari kedua metode tersebut [3].

Hybrid filtering memiliki kelebihan yang sama dengan collaborative filtering dan content-based filtering, seperti dapat memberikan rekomendasi yang tepat sesuai dengan preferensi pengguna, dan tidak tergantung pada kualitas data yang tersedia. Namun, hybrid filtering juga memiliki kekurangan, seperti memerlukan data yang lebih banyak dan lebih kompleks, serta dapat menjadi lebih sulit untuk diimplementasikan dan dioptimalkan [3].

BAB 3
Tahapan Collaborative Filtering

Berikut ini adalah tahap-tahap memberikan rekomendasi menggunakan collaborative filtering [4],[9],[10]:

1. Kumpulkan dan praproses data: Kumpulkan data peringkat dari pengguna dan praproses untuk mendapatkan matriks berukuran mxn, di mana m adalah jumlah pengguna dan n adalah jumlah item. Setiap elemen dari matriks mewakili peringkat yang diberikan kepada item oleh pengguna (jika peringkat tidak tersedia, itu direpresentasikan sebagai nilai yang hilang).
2. Hitung kesamaan: Hitung kesamaan antara pengguna dan/atau item menggunakan ukuran kesamaan seperti koefisien korelasi Pearson atau kesamaan kosinus. Ini dapat dilakukan dengan menggunakan matriks berukuran mxm untuk pemfilteran kolaboratif berbasis pengguna, atau matriks berukuran nxn untuk pemfilteran kolaboratif berbasis item.
3. Membuat prediksi: Untuk setiap pengguna, gunakan matriks kesamaan dan peringkat dari pengguna atau item serupa untuk memprediksi peringkat yang akan diberikan pengguna untuk item yang belum mereka beri peringkat. Hal ini dapat dilakukan dengan menggunakan rata-rata tertimbang, dimana bobot adalah kesamaan antara pengguna atau item.
4. Buat rekomendasi: Urutkan peringkat yang diprediksi dalam urutan menurun dan kembalikan k item teratas sebagai rekomendasi untuk setiap pengguna.
5. Mengevaluasi kinerja: Gunakan metrik evaluasi seperti presisi, daya ingat, dan rata-rata kesalahan kuadrat untuk mengukur kinerja sistem pemfilteran kolaboratif dan menyempurnakan parameter (seperti jumlah pengguna atau item serupa yang perlu dipertimbangkan) jika perlu.

BAB 4

Implementasi Collaborative Filtering

Implementasi Collaborative Filtering dimulai dengan menerapkan dasar-dasar algoritma pemfilteran kolaboratif seperti pengarsipan kolaboratif berbasis pengguna yang juga dikenal sebagai pemfilteran kolaboratif pengguna-ke-pengguna dan pemfilteran kolaboratif berbasis item (penyaringan kolaboratif item-ke-item).

Contoh Sintax implementasi Collaborative Filtering pada python:

```
import numpy as np

def predict(R, S, u, i, k=10):
    # R is the rating matrix, S is the similarity matrix
    # u is the index of the user, i is the index of the item
    # k is the number of similar users or items to consider

    # Get the indices of the k most similar users or items
    neighbors = np.argpartition(S[u,:], -k)[-k:]

    # Calculate the weighted average of the ratings
    rating = np.sum(R[neighbors, i] * S[u, neighbors]) / np.sum(S[u, neighbors])

    return rating

def recommend(R, S, u, k=10):
    # R is the rating matrix, S is the similarity matrix
    # u is the index of the user, k is the number of recommendations to make

    # Initialize a list to store the recommendations
    recommendations = []

    # Get the indices of all the items that the user has not yet rated
    unrated_items = np.where(R[u,:] == 0)[0]

    # Calculate the predicted rating for each unrated item
    ratings = [predict(R, S, u, i, k) for i in unrated_items]

    # Sort the ratings in descending order
    sorted_ratings = sorted(zip(unrated_items, ratings), key=lambda x: x[1],
reverse=True)

    # Return the top k items with the highest predicted ratings
    for item, rating in sorted_ratings[:k]:
        recommendations.append(item)

    return recommendations
```

Sintax implementasi diatas mengasumsikan bahwa Anda telah menghitung matriks kesamaan S menggunakan ukuran kesamaan seperti koefisien korelasi Pearson atau kesamaan kosinus. Fungsi "memprediksi" menggunakan matriks kesamaan untuk memprediksi peringkat yang akan diberikan pengguna pada item tertentu, dan fungsi "merekomendasikan" menggunakan fungsi "prediksi" untuk membuat rekomendasi kepada pengguna.

Anda kemudian dapat menggunakan fungsi-fungsi ini sebagai berikut:

```
# Load the rating matrix and the similarity matrix
R = np.load('rating_matrix.npy')
S = np.load('similarity_matrix.npy')

# Make recommendations for user 0
recommendations = recommend(R, S, 0)
print(recommendations)
```

Hal diatas ini akan mencetak daftar indeks item yang direkomendasikan kepada pengguna. Anda kemudian dapat menggunakan indeks ini untuk mencari item yang sesuai di kumpulan data Anda dan menampilkannya kepada pengguna.

BAB 5

Data Exploration Movielen Latest Small

Pada Bab ini, hal yang dilakukan adalah mejelajahi dataset kecil dari movielen. Kumpulan data ini digunakan di seluruh repositori untuk membangun sistem rekomendasi pemfilteran kolaboratif.

1.1. Download dan Explore Data

Hal pertama yag dilakukan adalah melakukan import pada program python yang berfungsi untuk memanggil file dalam satu module yang berbeda dan beriteraksi langsung dengan system operasi.

```
#Hal pertama yag dilakukan adalah melakukan import pada program python yang
#berfungsi untuk memanggil file dalam satu module yang berbeda dan beriteraksi

#langsung dengan system operasi.
import os

if not (os.path.exists("recsys.zip") or os.path.exists("recsys")):
    !wget https://github.com/nzhinusoftcm/review-on-collaborative-
          filtering/raw/master/recsys.zip
    !unzip recsys.zip
```

Hasil yang diperoleh, yaitu:

```
Location: https://raw.githubusercontent.com/nzhinusoftcm/review-on-collaborative-filtering/master/recsys.zip [following]
--2023-01-03 06:22:12--  https://raw.githubusercontent.com/nzhinusoftcm/review-on-collaborative-filtering/master/recsys.zip
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.111.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 15312323 (15M) [application/zip]
Saving to: 'recsys.zip'

recsys.zip          100%[===================>]  14.60M  --.-KB/s    in 0.1s

2023-01-03 06:22:13 (131 MB/s) - 'recsys.zip' saved [15312323/15312323]

Archive:  recsys.zip
   creating: recsys/
  inflating: recsys/datasets.py
  inflating: recsys/preprocessing.py
  inflating: recsys/utils.py
  inflating: recsys/requirements.txt
   creating: recsys/.vscode/
  inflating: recsys/.vscode/settings.json
   creating: recsys/__pycache__/
  inflating: recsys/__pycache__/datasets.cpython-36.pyc
  inflating: recsys/__pycache__/datasets.cpython-37.pyc
  inflating: recsys/__pycache__/utils.cpython-36.pyc
  inflating: recsys/__pycache__/preprocessing.cpython-37.pyc
  inflating: recsys/__pycache__/datasets.cpython-38.pyc
  inflating: recsys/__pycache__/preprocessing.cpython-36.pyc
  inflating: recsys/__pycache__/preprocessing.cpython-38.pyc
   creating: recsys/memories/
  inflating: recsys/memories/ItemToItem.py
  inflating: recsys/memories/UserToUser.py
   creating: recsys/memories/__pycache__/
  inflating: recsys/memories/__pycache__/UserToUser.cpython-36.pyc
  inflating: recsys/memories/__pycache__/UserToUser.cpython-37.pyc
  inflating: recsys/memories/__pycache__/ItemToItem.cpython-37.pyc
  inflating: recsys/memories/__pycache__/user2user.cpython-36.pyc
  inflating: recsys/memories/__pycache__/ItemToItem.cpython-36.pyc
   creating: recsys/models/
  inflating: recsys/models/SVD.py
  inflating: recsys/models/MatrixFactorization.py
  inflating: recsys/models/ExplainableMF.py
  inflating: recsys/models/NonnegativeMF.py
   creating: recsys/models/__pycache__/
  inflating: recsys/models/__pycache__/SVD.cpython-36.pyc
  inflating: recsys/models/__pycache__/MatrixFactorization.cpython-37.pyc
  inflating: recsys/models/__pycache__/ExplainableMF.cpython-36.pyc
  inflating: recsys/models/__pycache__/ExplainableMF.cpython-37.pyc
  inflating: recsys/models/__pycache__/MatrixFactorization.cpython-36.pyc
   creating: recsys/metrics/
  inflating: recsys/metrics/EvaluationMetrics.py
   creating: recsys/img/
  inflating: recsys/img/MF-and-NNMF.png
  inflating: recsys/img/svd.png
  inflating: recsys/img/MF.png
   creating: recsys/predictions/
   creating: recsys/predictions/item2item/
   creating: recsys/weights/
   creating: recsys/weights/item2item/
   creating: recsys/weights/item2item/ml1m/
  inflating: recsys/weights/item2item/ml1m/similarities.npy
  inflating: recsys/weights/item2item/ml1m/neighbors.npy
   creating: recsys/weights/item2item/ml100k/
  inflating: recsys/weights/item2item/ml100k/similarities.npy
  inflating: recsys/weights/item2item/ml100k/neighbors.npy
```

Dalam hal ini, Persyaratan yang berlaku adalah:

```
matplotlib==3.2.2
numpy==1.18.1
pandas==1.0.5
python==3.6.10
scikit-learn==0.23.1
scipy==1.5.0
```

Selanjutnya, Import Library atau Package

```
#Import library atau Package
from recsys.datasets import mlLatestSmall
import matplotlib.pyplot as plt
import pandas as pd
```

```
import zipfile
import urllib.request
import sys
import os
```

Kemudian, mendownload data Rating Film

```
#Mendownload data rating film
ratings, movies = mlLatestSmall.load()
```

Apabila download berhasil maka aka nada tampilan seperti dibawah ini:

```
Download data 100.5%
Successfully downloaded ml-latest-small.zip 978202 bytes.
Unzipping the ml-latest-small.zip zip file ...
```

1.2. Visualisasi data

Pada bagian ini, kita akan melakukan visualisasi dari data yang diperoleh, yaitu:

1. Menampilkan 5 data teratas dari data rating film

```
#Menampilkan 5 rating teratas dari film
ratings.head()
```

Dengan hasil yang diperoleh, yaitu:

	userid	itemid	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

2. Menampilkan 5 data teratas dari data film

```
#Menampilkan 5 film teratas
movies.head()
```

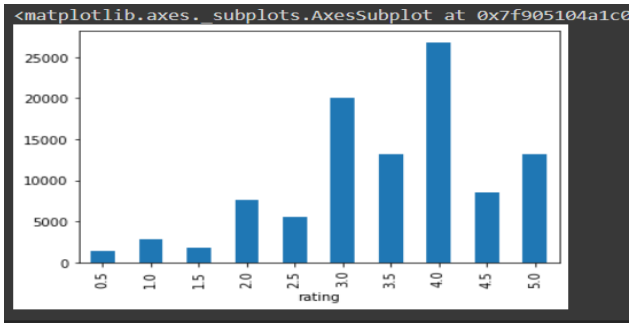
Hasil yang diperoleh, yaitu:

	itemid	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

3. Menampilkan Rating dalam bentuk Histogram

```
#Menunjukkan Histogram Dari rating film
ratings.groupby('rating').size().plot(kind='bar')
```

Hasil yang diperoleh, yaitu:

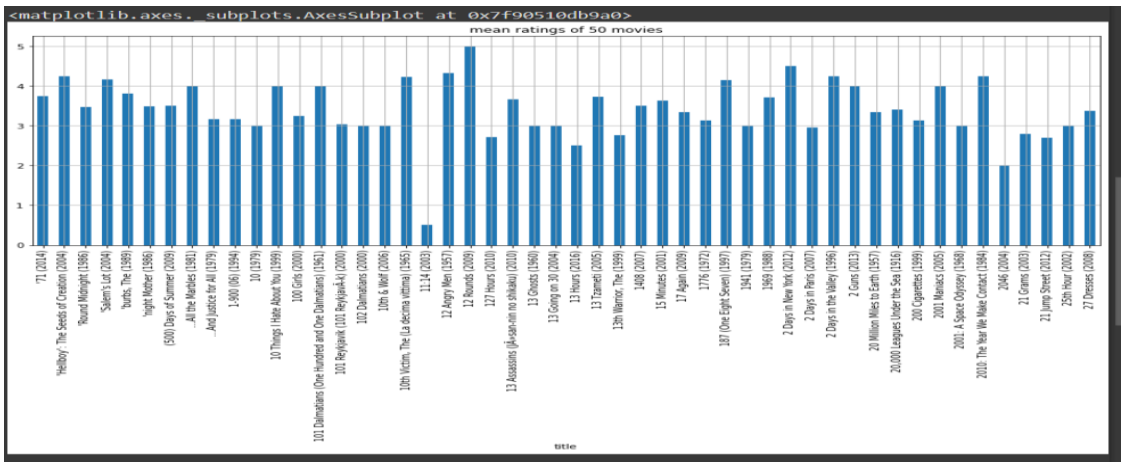


Histogram diatas menunjukkan bahwa Rating berkisar dari 0.5 hingga 0.5, dengan langkah atau jarak 0.5. Histogram di atas menampilkan partisi ulang peringkat dalam kumpulan data. dua peringkat yang paling umum adalah 4.0 dan 3.0 dan peringkat yang kurang umum adalah 0.5 dan 1.5

4. Menghitung Rata-rata film

```
#Menghitung rating rata-rata dari film
movie_means = ratings.join(movies['title'], on='itemid').groupby('titl
          e').rating.mean()
movie_means[:50].plot(kind='bar', grid=True, figsize=(16,6), title="me
          an ratings of 50 movies")
```

Hasil yang diperoleh:



5. Mengitung Perbadingan 30 film dengan rating tertinggi dan 30 film dengan rating lebih redah

```
#Menghitung Perbandingan 30 film dengan rating tertinggi dan 30 film
       dengan rating lebih redah
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(16,4), sharey=True)
movie_means.nlargest(30).plot(kind='bar', ax=ax1, title="Top 30 movie
       s in data set")
movie_means.nsmallest(30).plot(kind='bar', ax=ax2, title="Bottom 30 m
       ovies in data set")
```

Hasil yang diperoleh, yaitu:

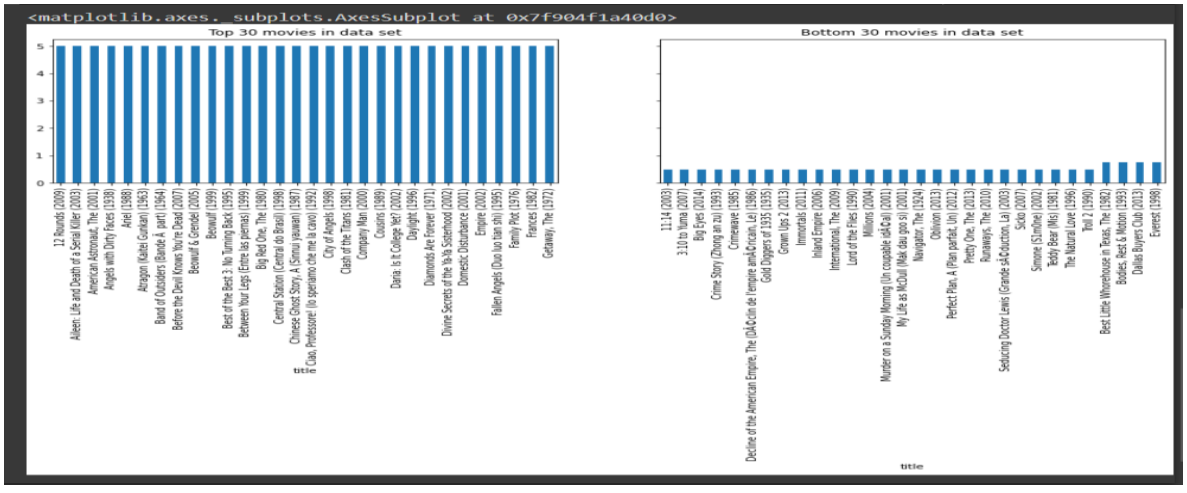


Diagram diatas menujukka bahwa, 30 data dengan rating tertinggi mencapai rata-rata 5 dan 30 data dengan rating lebih rendah memiliki rata-rata 1.

Kesimpulan:

Kesimpulan atau gagasan yang diperoleh di balik algoritma penyaringan kolaboratif berbasis pengguna adalah jika dua pengguna X dan Y memiliki perilaku serupa pada sekumpulan item, maka X kemungkinan besar akan memiliki perilaku serupa > ke Y pada item yang tidak berinteraksi dengannya.

BAB 6

Memory-based Collaborative Filtering

6.1. Definisi Memory-based Collaborative Filtering

Memory-based Collaborative Filtering adalah salah satu teknik yang digunakan dalam sistem rekomendasi untuk menghasilkan rekomendasi berdasarkan pada sejarah interaksi pengguna dengan item. Teknik ini menggunakan memori pengguna dan item untuk menemukan item yang paling mirip dengan item yang sedang dilihat pengguna. Item yang paling mirip tersebut kemudian digunakan untuk memprediksi apakah pengguna akan menyukai item tersebut atau tidak.

Ada dua jenis utama dari Memory-based Collaborative Filtering: user-based dan item-based. Pada user-based Collaborative Filtering, rekomendasi dibuat dengan mencari pengguna yang paling mirip dengan pengguna yang sedang dilihat, kemudian menyarankan item yang disukai oleh pengguna mirip tersebut tapi belum dilihat oleh pengguna yang sedang dilihat. Pada item-based Collaborative Filtering, rekomendasi dibuat dengan mencari item yang paling mirip dengan item yang sedang dilihat, kemudian menyarankan item yang mirip tersebut kepada pengguna yang sedang dilihat.

Memory-based Collaborative Filtering merupakan salah satu teknik yang cukup sederhana, namun cukup efektif dalam menghasilkan rekomendasi yang bermutu. Namun, teknik ini juga memiliki beberapa kelemahan, seperti membutuhkan banyak memori untuk menyimpan sejarah interaksi pengguna dengan item, dan juga tidak mampu menangani sparsity yang tinggi dalam data.

6.2. Pemfilteran kolaboratif berbasis memori

Pemfilteran kolaboratif berbasis memori (CF) juga dikenal sebagai CF berbasis tetangga terdekat membuat rekomendasi berdasarkan perilaku pengguna dan item yang serupa. Ada dua jenis CF berbasis memori: CF berbasis pengguna dan CF berbasis item. Kedua algoritma ini biasanya berjalan dalam tiga tahap:

a. Komputasi kesamaan (antara pengguna atau item)
b. Prediksi peringkat (menggunakan peringkat pengguna atau item serupa)
c. Rekomendasi Top-N

6.2.1. User-based Collaborative Filtering

Pemfilteran kolaboratif berbasis pengguna adalah teknik rekomendasi yang menggunakan peringkat pengguna serupa untuk membuat rekomendasi. Ini didasarkan pada gagasan bahwa pengguna yang memiliki selera yang sama pada item cenderung memiliki peringkat yang sama untuk item tersebut. Untuk menerapkan pemfilteran

kolaboratif berbasis pengguna, Anda memerlukan matriks peringkat R dengan ukuran m x n, di mana m adalah jumlah pengguna dan n adalah jumlah item. Setiap elemen R[i,j] mewakili peringkat yang diberikan oleh pengguna i ke item j (jika peringkat tidak tersedia, itu direpresentasikan sebagai nilai yang hilang).

Anda juga perlu menghitung kesamaan antara pengguna menggunakan ukuran kesamaan seperti koefisien korelasi Pearson atau kesamaan kosinus. Ini dapat dilakukan dengan menggunakan matriks S dengan ukuran m x m, di mana S[i,j] merepresentasikan kesamaan antara pengguna i dan j. Untuk membuat rekomendasi kepada pengguna u, Anda dapat menggunakan peringkat dari pengguna yang paling mirip dengan Anda untuk memprediksi peringkat yang akan Anda berikan pada item i. Ini dapat dilakukan sebagai rata-rata tertimbang dari peringkat pengguna serupa, di mana bobotnya adalah kesamaan di antara pengguna.

Misalnya, jika pengguna u memberi peringkat item 1 dan 2 dengan peringkat 4 dan 5, dan pengguna v memberi peringkat item 1 dan 2 dengan peringkat 3 dan 4, maka prediksi peringkat pengguna u untuk item 1 adalah (41 + 31) /(1+1) = 3,5. Anda kemudian dapat mengurutkan peringkat yang diprediksi dalam urutan menurun dan menampilkan k item teratas sebagai rekomendasi untuk pengguna.

6.2. 2. Definisi Item-to-item Collaborative Filtering

Pemfilteran kolaboratif item-ke-item adalah teknik rekomendasi yang menggunakan peringkat item serupa untuk membuat rekomendasi. Ini didasarkan pada gagasan bahwa item yang diberi peringkat serupa oleh pengguna cenderung terkait atau memiliki minat yang sama. Untuk menerapkan pemfilteran kolaboratif item-ke-item, Anda memerlukan matriks peringkat R dengan ukuran m x n, di mana m adalah jumlah pengguna dan n adalah jumlah item. Setiap elemen R[i,j] mewakili peringkat yang diberikan oleh pengguna i ke item j (jika peringkat tidak tersedia, itu direpresentasikan sebagai nilai yang hilang).

Dalamhal ini, perlu menghitung kesamaan antar item menggunakan ukuran kesamaan seperti koefisien korelasi Pearson atau kesamaan kosinus. Hal ini dapat dilakukan dengan menggunakan matriks P berukuran n x n, di mana P[i,j] merepresentasikan kesamaan antara item i dan j. Untuk membuat rekomendasi kepada pengguna u, Anda dapat menggunakan peringkat item serupa untuk memprediksi

peringkat yang akan Anda berikan pada item i. Ini dapat dilakukan sebagai rata-rata tertimbang dari peringkat item serupa, di mana bobotnya adalah kemiripan antar item.

Misalnya, jika pengguna u telah menilai item 1 dan 2 dengan peringkat 4 dan 5, dan item 1 serupa dengan item 3 dengan kemiripan 0,8, maka perkiraan peringkat pengguna u untuk item 3 adalah (4*0,8)/ 0,8 = 4. Anda kemudian dapat mengurutkan peringkat yang diprediksi dalam urutan menurun dan menampilkan k item teratas sebagai rekomendasi untuk pengguna.

6.2.3. Algoritma collaborative filtering

Algoritma yang digunakan pada user-based collaborative filtering, yaitu:

1. Input: matriks R berukuran m x n, dengan m adalah jumlah pengguna dan n adalah jumlah item. Setiap elemen R[i,j] merepresentasikan rating yang diberikan kepada item j oleh pengguna i (jika sebuah rating tidak tersedia, maka direpresentasikan sebagai missing value).
2. Inisialisasi matriks S berukuran mxm untuk menyimpan kesamaan antar pengguna.
3. Untuk setiap pengguna i dan j, hitung kesamaan antara pengguna i dan j menggunakan ukuran kesamaan seperti koefisien korelasi Pearson atau kesamaan kosinus. Simpan hasilnya di S[i,j].
4. Untuk setiap pengguna u, lakukan hal berikut:
    a. Inisialisasi daftar kosong R' untuk menyimpan item yang direkomendasikan untuk pengguna u.\
    b. Untuk setiap item i yang belum diberi peringkat oleh pengguna u, lakukan hal berikut:
        1. Hitung prediksi peringkat yang akan diberikan pengguna u ke item i menggunakan peringkat pengguna yang paling mirip dengan u. Ini dapat dilakukan sebagai rata-rata tertimbang, di mana bobotnya adalah kesamaan antara pengguna.
        2. Tambahkan peringkat yang diprediksi dan item i ke R'.
    c. Urutkan R' dalam urutan menurun berdasarkan peringkat yang diprediksi dan kembalikan k item teratas sebagai rekomendasi untuk pengguna u.

Algoritma ini menggunakan pemfilteran kolaboratif berbasis pengguna untuk membuat rekomendasi kepada setiap pengguna. Ini pertama-tama menghitung kesamaan antara semua pasangan pengguna, dan kemudian menggunakan peringkat dari pengguna yang paling mirip untuk membuat prediksi untuk setiap pengguna.

6.3. Implementasi User-based Collaborative Filtering

Hal yang dilakukan adalah:

1. Mendownload tools

```
#Mendowload tools
import os

if not (os.path.exists("recsys.zip") or os.path.exists("recsys")):
    !wget https://github.com/nzhinusoftcm/review-on-collaborative-filtering/raw/master/recsys.zip
    !unzip recsys.zip
```

Hasil yang diperoleh, yaitu:

```
--2023-01-02 12:43:29--  https://github.com/nzhinusoftcm/review-on-collaborative-filtering/raw/master/recsys.zip
Resolving github.com (github.com)... 140.82.114.4
Connecting to github.com (github.com)|140.82.114.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/nzhinusoftcm/review-on-collaborative-filtering/master/recsys.zip [following]
--2023-01-02 12:43:30--  https://raw.githubusercontent.com/nzhinusoftcm/review-on-collaborative-filtering/master/recsys.zip
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 15312323 (15M) [application/zip]
Saving to: 'recsys.zip'

recsys.zip          100%[===================>]  14.60M  --.-KB/s    in 0.1s

2023-01-02 12:43:30 (125 MB/s) - 'recsys.zip' saved [15312323/15312323]

Archive:  recsys.zip
   creating: recsys/
  inflating: recsys/datasets.py
  inflating: recsys/preprocessing.py
  inflating: recsys/utils.py
  inflating: recsys/requirements.txt
   creating: recsys/.vscode/
  inflating: recsys/.vscode/settings.json
   creating: recsys/__pycache__/
  inflating: recsys/__pycache__/datasets.cpython-36.pyc
  inflating: recsys/__pycache__/datasets.cpython-37.pyc
  inflating: recsys/__pycache__/utils.cpython-36.pyc
  inflating: recsys/__pycache__/preprocessing.cpython-37.pyc
  inflating: recsys/__pycache__/datasets.cpython-38.pyc
  inflating: recsys/__pycache__/preprocessing.cpython-36.pyc
  inflating: recsys/__pycache__/preprocessing.cpython-38.pyc
   creating: recsys/memories/
  inflating: recsys/memories/ItemToItem.py
  inflating: recsys/memories/UserToUser.py
   creating: recsys/memories/__pycache__/
  inflating: recsys/memories/__pycache__/UserToUser.cpython-36.pyc
  inflating: recsys/memories/__pycache__/UserToUser.cpython-37.pyc
  inflating: recsys/memories/__pycache__/ItemToItem.cpython-37.pyc
  inflating: recsys/memories/__pycache__/user2user.cpython-36.pyc
  inflating: recsys/memories/__pycache__/ItemToItem.cpython-36.pyc
   creating: recsys/models/
  inflating: recsys/models/SVD.py
  inflating: recsys/models/MatrixFactorization.py
  inflating: recsys/models/ExplainableMF.py
  inflating: recsys/models/NonnegativeMF.py
```

Dengan:

```
Import Persyaratan
matplotlib==3.2.2
numpy==1.19.2
pandas==1.0.5
```

```
python==3.7
scikit-learn==0.24.1
scikit-surprise==1.1.1
scipy==1.6.2
```

2. Import Library atau Package

```
#import Library atau Package
from sklearn.neighbors import NearestNeighbors
from scipy.sparse import csr_matrix

from recsys.datasets import ml100k
from recsys.preprocessing import ids_encoder

import pandas as pd
import numpy as np
import zipfile
```

3. Memuat atau mendownload Rating Movielen

```
#Mendownload Rating Movielen
ratings, movies = ml100k.load()
```

Hasil ysng diperoleh, yaitu:

```
Download data 100.2%
Successfully downloaded ml-100k.zip 4924029 bytes.
Unzipping the ml-100k.zip zip file ...
```

4. Pengkodean Userid dan Itemid

```
# create the encoder
ratings, uencoder, iencoder = ids_encoder(ratings)
```

5. Ubah Dataframe peringkat menjadi matriks

```
#Ubah Dataframe peringkat menjadi matriks
def ratings_matrix(ratings):
    return csr_matrix(pd.crosstab(ratings.userid, ratings.itemid, ratin
gs.rating, aggfunc=sum).fillna(0).values)

R = ratings_matrix(ratings)
```

6. Mengembalikan knn pengguna untuk setiap pengguna

```
#mengembalikan knn pengguna untuk setiap pengguna.
```

```
def create_model(rating_matrix, metric):
    """
    - create the nearest neighbors model with the corresponding similar
ity metric
    - fit the model
    """
    model = NearestNeighbors(metric=metric, n_neighbors=21, algorithm='
brute')
    model.fit(rating_matrix)
    return model
```

```
#mengembalikan knn pengguna untuk setiap pengguna.
def create_model(rating_matrix, metric):
    """
    - create the nearest neighbors model with the corresponding similar
ity metric
    - fit the model
    """
    model = NearestNeighbors(metric=metric, n_neighbors=21, algorithm='
brute')
    model.fit(rating_matrix)
    return model
def nearest_neighbors(rating_matrix, model):
    """
    :param rating_matrix : rating matrix of shape (nb_users, nb_items)
    :param model : nearest neighbors model
    :return
        - similarities : distances of the neighbors from the referenced
 user
        - neighbors : neighbors of the referenced user in decreasing or
der of similarities
    """
    similarities, neighbors = model.kneighbors(rating_matrix)
    return similarities[:, 1:], neighbors[:, 1:]
```

7. Memanggil fungsi create_model() dan neighbor_neighbors() masing-masing untuk membuat model -NN dan menghitung tetangga terdekat untuk pengguna tertentu

```
#Memanggil fungsi create_model() dan neighbor_neighbors() masing-
masing untuk membuat
#model -NN dan menghitung tetangga terdekat untuk pengguna tertentu

model = create_model(rating_matrix=R, metric='cosine') # we can also us
e the 'euclidian' distance
similarities, neighbors = nearest_neighbors(R, model)
```

8. Menemukan item yang dibeli oleh pengguna serupa ini beserta frekuensinya

```
#Menemukan item yang dibeli oleh pengguna serupa ini beserta frekuensin
ya
def find_candidate_items(userid):
    """
    Find candidate items for an active user

    :param userid : active user
    :param neighbors : users similar to the active user
    :return candidates : top 30 of candidate items
    """
    user_neighbors = neighbors[userid]
    activities = ratings.loc[ratings.userid.isin(user_neighbors)]

    # sort items in decreasing order of frequency
    frequency = activities.groupby('itemid')['rating'].count().reset_in
dex(name='count').sort_values(['count'],ascending=False)
    Gu_items = frequency.itemid
    active_items = ratings.loc[ratings.userid == userid].itemid.to_list
()
    candidates = np.setdiff1d(Gu_items, active_items, assume_unique=Tru
e)[:30]

    return candidates
```

9. Selanjutnya, hitung nilai rata-rata setiap pengguna dan nilai yang dinormalisasi untuk setiap item. Rata-rata DataFrame berisi nilai rata-rata untuk setiap pengguna. Dengan rating rata-rata setiap pengguna, dapat menambahkan kolom tambahan norm_rating ke DataFrame rating yang dapat diakses untuk membuat prediksi.

```
# mean ratings for each user
mean = ratings.groupby(by='userid', as_index=False)['rating'].mean()
mean_ratings = pd.merge(ratings, mean, suffixes=('','_mean'), on='useri
d')

# normalized ratings for each items
mean_ratings['norm_rating'] = mean_ratings['rating'] - mean_ratings['ra
ting_mean']

mean = mean.to_numpy()[:, 1]
```

10. Definisikan fungsi predict yang memprediksi rating antara user u dan item i .

```
#definisikan fungsi predict yang memprediksi rating antara user u dan i
tem i .
def predict(userid, itemid):
    """
```

```
    predict what score userid would have given to itemid.

    :param
        - userid : user id for which we want to make prediction
        - itemid : item id on which we want to make prediction

    :return
        - r_hat : predicted rating of user userid on item itemid
    """
    user_similarities = similarities[userid]
    user_neighbors = neighbors[userid]
    # get mean rating of user userid
    user_mean = mean[userid]

    # find users who rated item 'itemid'
    iratings = np_ratings[np_ratings[:, 1].astype('int') == itemid]

    # find similar users to 'userid' who rated item 'itemid'
    suri = iratings[np.isin(iratings[:, 0], user_neighbors)]

    # similar users who rated current item (surci)
    normalized_ratings = suri[:,4]
    indexes = [np.where(user_neighbors == uid)[0][0] for uid in suri[:,
 0].astype('int')]
    sims = user_similarities[indexes]

    num = np.dot(normalized_ratings, sims)
    den = np.sum(np.abs(sims))

    if num == 0 or den == 0:
        return user_mean

    r_hat = user_mean + np.dot(normalized_ratings, sims) / np.sum(np.ab
s(sims))

    return r_hat
```

11. Membuat prediksi peringkat untuk pengguna tertentu pada setiap item dalam rangkaian item kandidatnya.

```
#Membuat prediksi peringkat untuk pengguna tertentu pada setiap item da
lam rangkaian item kandidatnya.
def user2userPredictions(userid, pred_path):
    """
    Make rating prediction for the active user on each candidate item a
nd save in file prediction.csv

    :param
```

```
        - userid : id of the active user
        - pred_path : where to save predictions
    """
    # find candidate items for the active user
    candidates = find_candidate_items(userid)

    # loop over candidates items to make predictions
    for itemid in candidates:

        # prediction for userid on itemid
        r_hat = predict(userid, itemid)

        # save predictions
        with open(pred_path, 'a+') as file:
            line = '{},{},{}\n'.format(userid, itemid, r_hat)
            file.write(line)
```

```
#hitung rating prediksi

import sys


def user2userCF():
    """
    Make predictions for each user in the database.
    """
    # get list of users in the database
    users = ratings.userid.unique()

    def _progress(count):
        sys.stdout.write('\rRating predictions. Progress status : %.1f%
%' % (float(count/len(users))*100.0))
        sys.stdout.flush()

    saved_predictions = 'predictions.csv'
    if os.path.exists(saved_predictions):
        os.remove(saved_predictions)

    for count, userid in enumerate(users):
        # make rating predictions for the current user
        user2userPredictions(userid, saved_predictions)
        _progress(count)
```

```
#hitung rating prediksi
user2userCF()
```

12. Membaca prediksi untuk pengguna tertentu dan mengembalikan daftar item dalam urutan menurun dari peringkat yang diprediksi.

```
#membaca prediksi untuk pengguna tertentu dan mengembalikan daftar item
 dalam urutan menurun dari peringkat yang diprediksi.
def user2userRecommendation(userid):
    """
    """
    # encode the userid
    uid = uencoder.transform([userid])[0]
    saved_predictions = 'predictions.csv'

    predictions = pd.read_csv(saved_predictions, sep=',', names=['useri
d', 'itemid', 'predicted_rating'])
    predictions = predictions[predictions.userid==uid]
    List = predictions.sort_values(by=['predicted_rating'], ascending=F
alse)

    List.userid = uencoder.inverse_transform(List.userid.tolist())
    List.itemid = iencoder.inverse_transform(List.itemid.tolist())

    List = pd.merge(List, movies, on='itemid', how='inner')

    return List
```

```
#membaca data itemid 212
user2userRecommendation(212)
```

Hasil yang diperoleh, yaitu:

	userid	itemid	predicted_rating	title
0	212	483	4.871495	Casablanca (1942)
1	212	357	4.764547	One Flew Over the Cuckoo's Nest (1975)
2	212	50	4.660002	Star Wars (1977)
3	212	98	4.613636	Silence of the Lambs, The (1991)
4	212	64	4.550733	Shawshank Redemption, The (1994)
5	212	194	4.522336	Sting, The (1973)
6	212	174	4.521300	Raiders of the Lost Ark (1981)
7	212	134	4.414819	Citizen Kane (1941)
8	212	187	4.344531	Godfather: Part II, The (1974)
9	212	196	4.303696	Dead Poets Society (1989)
10	212	523	4.281802	Cool Hand Luke (1967)
11	212	216	4.278246	When Harry Met Sally... (1989)
12	212	100	4.260087	Fargo (1996)
13	212	168	4.206139	Monty Python and the Holy Grail (1974)

13. Evaluasi dengan Mean Absolute Error (MAE)

```
#Evaluasi dengan Mean Absolute Error (MAE)
from recsys.preprocessing import train_test_split, get_examples

# get examples as tuples of userids and itemids and labels from normali
ze ratings
raw_examples, raw_labels = get_examples(ratings, labels_column='rating'
)

# train test split
(x_train, x_test), (y_train, y_test) = train_test_split(examples=raw_ex
amples, labels=raw_labels)

def evaluate(x_test, y_test):
    print('Evaluate the model on {} test data ...'.format(x_test.shape[
0]))
    preds = list(predict(u,i) for (u,i) in x_test)
    mae = np.sum(np.absolute(y_test - np.array(preds))) / x_test.shape[
0]
    print('\nMAE :', mae)
    return mae
```

```
#Evaluasi nilai MAE
evaluate(x_test, y_test)
```

Hasil yang diperoleh, yaitu:

```
Evaluate the model on 10000 test data ...

MAE : 0.7505910931068639
0.7505910931068639
```

14. Evaluasi pada set data ML-100k

```
#Evaluasi pada set data ML-100k
from recsys.memories.UserToUser import UserToUser

# load ml100k ratings
ratings, movies = ml100k.load()

# prepare data
ratings, uencoder, iencoder = ids_encoder(ratings)

# get examples as tuples of userids and itemids and labels from normali
ze ratings
raw_examples, raw_labels = get_examples(ratings, labels_column='rating'
)

# train test split
(x_train, x_test), (y_train, y_test) = train_test_split(examples=raw_ex
amples, labels=raw_labels)
```

15. Membangun user-Based CF

```
# create the user-based CF
usertouser = UserToUser(ratings, movies, metric='cosine')
```

Hasil yang diperoleh, yaitu:

```
Normalize users ratings ...
Initialize the similarity model ...
Compute nearest neighbors ...
User to user recommendation model created with success ...
```

16. Evaluasi user-Based CF

```
# evaluate the user-based CF on the ml100k test data
usertouser.evaluate(x_test, y_test)
```

Hasil yang diperoleh, yaitu:

```
Evaluate the model on 10000 test data ...

MAE : 0.7505910931068639
0.7505910931068639
```

17. Evaluasi pada kumpulan data ML-1M

```
from recsys.datasets import ml1m
from recsys.preprocessing import ids_encoder, get_examples, train_test_
split
from recsys.memories.UserToUser import UserToUser

# load ml100k ratings
ratings, movies = ml1m.load()

# prepare data
ratings, uencoder, iencoder = ids_encoder(ratings)

# get examples as tuples of userids and itemids and labels from normali
ze ratings
raw_examples, raw_labels = get_examples(ratings, labels_column='rating'
)

# train test split
(x_train, x_test), (y_train, y_test) = train_test_split(examples=raw_ex
amples, labels=raw_labels)

# create the user-based CF
usertouser = UserToUser(ratings, movies, k=20, metric='cosine')

# evaluate the user-based CF on the ml1m test data
print("============================")
usertouser.evaluate(x_test, y_test)
```

Hasil yang diperoleh:

```
Download data 100.1%
Successfully downloaded ml-1m.zip 5917549 bytes.
Unzipping the ml-1m.zip zip file ...
Normalize users ratings ...
Initialize the similarity model ...
Compute nearest neighbors ...
User to user recommendation model created with success ...
============================
Evaluate the model on 100021 test data ...

MAE : 0.732267005840993
0.732267005840993
```

6.4. Implementasi Item-to-Item Collaborative Filtering

1. Hal pertama yag dilakukan adalah melakukan import pada program python yang berfungsi untuk memanggil file dalam satu module yang berbeda dan beriteraksi langsung dengan system operasi.

```
#Hal pertama yag dilakukan adalah melakukan import pada program python
            yang
#berfungsi untuk memanggil file dalam satu module yang berbeda dan ber
           iteraksi
#langsung dengan system operasi.
import os

if not (os.path.exists("recsys.zip") or os.path.exists("recsys")):
    !wget https://github.com/nzhinusoftcm/review-on-collaborative-
          filtering/raw/master/recsys.zip
    !unzip recsys.zip
```

Hasil yang diperoleh, yaitu:

```
--2023-01-02 13:10:43--  https://github.com/nzhinusoftcm/review-on-collaborative-filtering/raw/mas
Resolving github.com (github.com)... 140.82.113.4
Connecting to github.com (github.com)|140.82.113.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/nzhinusoftcm/review-on-collaborative-filtering/master/
--2023-01-02 13:10:43--  https://raw.githubusercontent.com/nzhinusoftcm/review-on-collaborative-fi
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.13
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connec
HTTP request sent, awaiting response... 200 OK
Length: 15312323 (15M) [application/zip]
Saving to: 'recsys.zip'

recsys.zip          100%[===================>]  14.60M  --.-KB/s    in 0.04s

2023-01-02 13:10:43 (364 MB/s) - 'recsys.zip' saved [15312323/15312323]

Archive:  recsys.zip
   creating: recsys/
  inflating: recsys/datasets.py
  inflating: recsys/preprocessing.py
  inflating: recsys/utils.py
  inflating: recsys/requirements.txt
   creating: recsys/.vscode/
  inflating: recsys/.vscode/settings.json
   creating: recsys/__pycache__/
  inflating: recsys/__pycache__/datasets.cpython-36.pyc
  inflating: recsys/__pycache__/datasets.cpython-37.pyc
  inflating: recsys/__pycache__/utils.cpython-36.pyc
  inflating: recsys/__pycache__/preprocessing.cpython-37.pyc
  inflating: recsys/__pycache__/datasets.cpython-38.pyc
  inflating: recsys/__pycache__/preprocessing.cpython-36.pyc
  inflating: recsys/__pycache__/preprocessing.cpython-38.pyc
   creating: recsys/memories/
  inflating: recsys/memories/ItemToItem.py
  inflating: recsys/memories/UserToUser.py
   creating: recsys/memories/__pycache__/
  inflating: recsys/memories/__pycache__/UserToUser.cpython-36.pyc
  inflating: recsys/memories/__pycache__/UserToUser.cpython-37.pyc
  inflating: recsys/memories/__pycache__/ItemToItem.cpython-37.pyc
  inflating: recsys/memories/__pycache__/user2user.cpython-36.pyc
  inflating: recsys/memories/__pycache__/ItemToItem.cpython-36.pyc
   creating: recsys/models/
  inflating: recsys/models/SVD.py
  inflating: recsys/models/MatrixFactorization.py
  inflating: recsys/models/ExplainableMF.py
  inflating: recsys/models/NonnegativeMF.py
```

Dengan Persyaratan yang berlaku adalah:

```
matplotlib==3.2.2
```

```
numpy==1.18.1
pandas==1.0.5
python==3.6.10
scikit-learn==0.23.1
scipy==1.5.0
```

2. Import Library atau Package

```
#Import Library atau Package
from sklearn.neighbors import NearestNeighbors
from scipy.sparse import csr_matrix

from recsys.datasets import ml1m, ml100k
from recsys.preprocessing import ids_encoder

import pandas as pd
import numpy as np
import os
import sys
```

3. Download data yang memuat Ratings

```
#Download data yang memuat Ratings
ratings, movies = ml100k.load()
```

Hasil yang diperoleh, yaitu:

```
Download data 100.2%
Successfully downloaded ml-100k.zip 4924029 bytes.
Unzipping the ml-100k.zip zip file ...
```

4. Pengodean userid dan itemid

```
# create the encoder
ratings, uencoder, iencoder = ids_encoder(ratings)
```

5. Temukan kesamaan untuk setiap item

```
#Temukan kesamaan untuk setiap item
def normalize():
    # compute mean rating for each user
    mean = ratings.groupby(by='userid', as_index=False)['rating'].mean(
)
```

```
    norm_ratings = pd.merge(ratings, mean, suffixes=('','_mean'), on='u
serid')

    # normalize each rating by substracting the mean rating of the corr
esponding user
    norm_ratings['norm_rating'] = norm_ratings['rating'] - norm_ratings
['rating_mean']
    return mean.to_numpy()[:, 1], norm_ratings
```

```
#peringkat telah dinormalisasi
mean, norm_ratings = normalize()
np_ratings = norm_ratings.to_numpy()
norm_ratings.head()
```

Hasil yang diperoleh:

	userid	itemid	rating	rating_mean	norm_rating
0	0	0	5	3.610294	1.389706
1	0	1	3	3.610294	-0.610294
2	0	2	4	3.610294	0.389706
3	0	3	3	3.610294	-0.610294
4	0	4	3	3.610294	-0.610294

6. setiap item mewakili dengan vektor dari peringkat yang dinormalisasi

```
#setiap item mewakili dengan vektor dari peringkat yang dinormalisasi
def item_representation(ratings):
    return csr_matrix(
        pd.crosstab(ratings.itemid, ratings.userid, ratings.norm_rating
, aggfunc=sum).fillna(0).values
    )
```

```
#item representasi
R = item_representation(norm_ratings)
```

7. Membangun dan menyesuaikan model -NN menggunakan sklearn

```
#membangun dan menyesuaikan model -NN menggunakan sklearn
def create_model(rating_matrix, k=20, metric="cosine"):
    """
```

```
    :param R : numpy array of item representations
    :param k : number of nearest neighbors to return
    :return model : our knn model
    """
    model = NearestNeighbors(metric=metric, n_neighbors=k+1, algorithm=
'brute')
    model.fit(rating_matrix)
    return model
```

8. Perhitungan persamaan Kesamaan antar item dapat diukur dengan jarak Cosine atau Eucliedian. Kelas *NearestNeighbors* dari perpustakaan sklearn menyederhanakan perhitungan tetangga.

```
#Perhitungan persamaan Kesamaan antar item dapat diukur dengan jarak Co
sine atau Eucliedian
def nearest_neighbors(rating_matrix, model):
    """
    compute the top n similar items for each item.
    :param rating_matrix : items representations
    :param model : nearest neighbors model
    :return similarities, neighbors
    """
    similarities, neighbors = model.kneighbors(rating_matrix)
    return similarities[:,1:], neighbors[:,1:]
```

9. Kesamaan Kosinus yang Disesuaikan

```
#Kesamaan Kosinus yang Disesuaikan
def save_similarities(similarities, neighbors, dataset_name):
    base_dir = 'recsys/weights/item2item'
    save_dir = os.path.join(base_dir, dataset_name)
    os.makedirs(save_dir, exist_ok=True)
    similarities_file_name = os.path.join(save_dir, 'similarities.npy')
    neighbors_file_name = os.path.join(save_dir, 'neighbors.npy')
    try:
        np.save(similarities_file_name, similarities)
        np.save(neighbors_file_name, neighbors)
    except ValueError as error:
        print(f"An error occured when saving similarities, due to : \n
ValueError : {error}")


def load_similarities(dataset_name, k=20):
    base_dir = 'recsys/weights/item2item'
    save_dir = os.path.join(base_dir, dataset_name)
    similiraties_file = os.path.join(save_dir, 'similarities.npy')
```

```
    neighbors_file = os.path.join(save_dir, 'neighbors.npy')
    similarities = np.load(similiraties_file)
    neighbors = np.load(neighbors_file)
    return similarities[:,:k], neighbors[:,:k]


def cosine(x, y):
    return np.dot(x, y) / (np.linalg.norm(x) * np.linalg.norm(y))


def adjusted_cosine(np_ratings, nb_items, dataset_name):
    similarities = np.zeros(shape=(nb_items, nb_items))
    similarities.fill(-1)

    def _progress(count):
        sys.stdout.write('\rComputing similarities. Progress status : %
.1f%%' % (float(count / nb_items)*100.0))
        sys.stdout.flush()

    items = sorted(ratings.itemid.unique())
    for i in items[:-1]:
        for j in items[i+1:]:
            scores = np_ratings[(np_ratings[:, 1] == i) | (np_ratings[:
, 1] == j), :]
            vals, count = np.unique(scores[:,0], return_counts = True)
            scores = scores[np.isin(scores[:,0], vals[count > 1]),:]

            if scores.shape[0] > 2:
                x = scores[scores[:, 1].astype('int') == i, 4]
                y = scores[scores[:, 1].astype('int') == j, 4]
                w = cosine(x, y)

                similarities[i, j] = w
                similarities[j, i] = w
        _progress(i)
    _progress(nb_items)

    # get neighbors by their neighbors in decreasing order of similarit
ies
    neighbors = np.flip(np.argsort(similarities), axis=1)

    # sort similarities in decreasing order
    similarities = np.flip(np.sort(similarities), axis=1)

    # save similarities to disk
    save_similarities(similarities, neighbors, dataset_name=dataset_nam
e)

    return similarities, neighbors
```

10. Batalkan komentar pada dua baris sel berikut untuk menghitung cosinus yang disesuaikan di antara semua item.

```
#batalkan komentar pada dua baris sel berikut untuk menghitung cosinus
yang disesuaikan
#di antara semua item.
# nb_items = ratings.itemid.nunique()
# similarities, neighbors = adjusted_cosine(np_ratings, nb_items=nb_ite
ms, dataset_name='ml100k'
```

11. menggunakan metrik yang disesuaikan_cosinus

```
# metric : choose among [cosine, euclidean, adjusted_cosine]

metric = 'adjusted_cosine'

if metric == 'adjusted_cosine':
    similarities, neighbors = load_similarities('ml100k')
else:
    model = create_model(R, k=21, metric=metric)
    similarities, neighbors = nearest_neighbors(R, model)
```

```
print('neighbors shape : ', neighbors.shape)
print('similarities shape : ', similarities.shape)
```

Hasil yang diperoleh:

```
neighbors shape :  (1682, 20)
similarities shape :  (1682, 20)
```

12. menemukan kandidat

```
#menemukan kandidat
def candidate_items(userid):
    """
    :param userid : user id for which we wish to find candidate items

    :return : I_u, candidates
    """

    # 1. Finding the set I_u of items already rated by user userid
    I_u = np_ratings[np_ratings[:, 0] == userid]
```

```
    I_u = I_u[:, 1].astype('int')

    # 2. Taking the union of similar items for all items in I_u to form
 the set of candidate items
    c = set()

    for iid in I_u:
        # add the neighbors of item iid in the set of candidate items
        c.update(neighbors[iid])

    c = list(c)
    # 3. exclude from the set C all items in I_u.
    candidates = np.setdiff1d(c, I_u, assume_unique=True)

    return I_u, candidates
```

```
test_user = uencoder.transform([1])[0]
i_u, u_candidates = candidate_items(test_user)
```

```
print('number of items purchased by user 1 : ', len(i_u))
print('number of candidate items for user 1 : ', len(u_candidates))
```

Hasil yang diperoleh:

```
number of items purchased by user 1 :  272
number of candidate items for user 1 :  893
```

13. Temukan kesamaan antara setiap kandidat item dan himpunan

```
#Temukan kesamaan antara setiap kandidat item dan himpunan
def similarity_with_Iu(c, I_u):
    """
    compute similarity between an item c and a set of items I_u. For ea
ch item i in I_u, get similarity between
    i and c, if c exists in the set of items similar to itemid.
    :param c : itemid of a candidate item
    :param I_u : set of items already purchased by a given user
    :return w : similarity between c and I_u
    """
    w = 0
    for iid in I_u :
        # get similarity between itemid and c, if c is one of the k nea
rest neighbors of itemid
        if c in neighbors[iid] :
```

```
            w = w + similarities[iid, neighbors[iid] == c][0]
    return w
```

14. Beri peringkat item kandidat berdasarkan kemiripannya

```
#Beri peringkat item kandidat berdasarkan kemiripannya
def rank_candidates(candidates, I_u):
    """
    rank candidate items according to their similarities with i_u
    :param candidates : list of candidate items
    :param I_u : list of items purchased by the user
    :return ranked_candidates : dataframe of candidate items, ranked in
 descending order of similarities with I_u
    """

    # list of candidate items mapped to their corresponding similaritie
s to I_u
    sims = [similarity_with_Iu(c, I_u) for c in candidates]
    candidates = iencoder.inverse_transform(candidates)
    mapping = list(zip(candidates, sims))

    ranked_candidates = sorted(mapping, key=lambda couple:couple[1], re
verse=True)
    return ranked_candidates
```

15. membuat rekomendasi top-N untuk pengguna tertentu

```
#membuat rekomendasi top-N untuk pengguna tertentu
def topn_recommendation(userid, N=30):
    """
    Produce top-N recommendation for a given user
    :param userid : user for which we produce top-N recommendation
    :param n : length of the top-N recommendation list
    :return topn
    """
    # find candidate items
    I_u, candidates = candidate_items(userid)

    # rank candidate items according to their similarities with I_u
    ranked_candidates = rank_candidates(candidates, I_u)

    # get the first N row of ranked_candidates to build the top N recom
mendation list
    topn = pd.DataFrame(ranked_candidates[:N], columns=['itemid','simil
arity_with_Iu'])
    topn = pd.merge(topn, movies, on='itemid', how='inner')
    return topn
```

```
#menampilkan rekomendasi top-N untuk pengguna tertentu
topn_recommendation(test_user)
```

Hasil yang diperoleh, yaitu:

	itemid	similarity_with_Iu	title
0	1356	52.867173	Ed's Next Move (1996)
1	1189	50.362199	Prefontaine (1997)
2	1516	31.133267	Wedding Gift, The (1994)
3	1550	31.031738	Destiny Turns on the Radio (1995)
4	1554	27.364494	Safe Passage (1994)
5	1600	27.287712	Guantanamera (1994)
6	1223	26.631850	King of the Hill (1993)
7	1388	26.624397	Gabbeh (1996)
8	766	26.590175	Man of the Year (1995)
9	691	26.461802	Dark City (1998)
10	1378	25.787842	Rhyme & Reason (1997)
11	1664	25.327445	8 Heads in a Duffel Bag (1997)
12	1261	24.785660	Run of the Country, The (1995)
13	1123	24.524028	Last Time I Saw Paris, The (1954)
14	1538	24.492453	All Over Me (1997)
15	1485	24.345312	Colonel Chabert, Le (1994)
16	1450	24.262120	Golden Earrings (1947)
17	909	23.357301	Dangerous Beauty (1998)
18	359	22.973658	Assignment, The (1997)
19	1369	22.710078	Forbidden Christ, The (Cristo proibito, Il) (1...

16. Prediksi Rating

```
#Prediksi Rating
def predict(userid, itemid):
    """
    Make rating prediction for user userid on item itemid
    :param userid : id of the active user
    :param itemid : id of the item for which we are making prediction

    :return r_hat : predicted rating
    """

    # Get items similar to item itemid with their corresponding similar
ities
    item_neighbors = neighbors[itemid]
    item_similarities = similarities[itemid]

    # get ratings of user with id userid
    uratings = np_ratings[np_ratings[:, 0].astype('int') == userid]

    # similar items rated by item the user of i
    siru = uratings[np.isin(uratings[:, 1], item_neighbors)]
    scores = siru[:, 2]
```

```
    indexes = [np.where(item_neighbors == iid)[0][0] for iid in siru[:,
1].astype('int')]
    sims = item_similarities[indexes]

    dot = np.dot(scores, sims)
    som = np.sum(np.abs(sims))

    if dot == 0 or som == 0:
        return mean[userid]

    return dot / som
```

17. memprediksi peringkat apa yang akan diberikan pengguna ke daftar top-N sebelumnya dan mengembalikan daftar yang ditata ulang (dalam urutan prediksi menurun) sebagai daftar top-N baru.

```
#memprediksi peringkat apa yang akan diberikan pengguna ke daftar top-
N sebelumnya
#dan mengembalikan daftar yang ditata ulang
def topn_prediction(userid):
    """
    :param userid : id of the active user
    :return topn : initial topN recommendations returned by the functio
n item2item_topN
    :return topn_predict : topN recommendations reordered according to
rating predictions
    """
    # make top N recommendation for the active user
    topn = topn_recommendation(userid)

    # get list of items of the top N list
    itemids = topn.itemid.to_list()

    predictions = []

    # make prediction for each item in the top N list
    for itemid in itemids:
        r = predict(userid, itemid)

        predictions.append((itemid,r))

    predictions = pd.DataFrame(predictions, columns=['itemid','predicti
on'])
```

```
    # merge the predictions to topN_list and rearrange the list accordi
ng to predictions
    topn_predict = pd.merge(topn, predictions, on='itemid', how='inner'
)
    topn_predict = topn_predict.sort_values(by=['prediction'], ascendin
g=False)

    return topn, topn_predict
```

18. buat rekomendasi untuk pengguna 1 dan bandingkan kedua daftar yang ada

```
#prediksi Top N
topn, topn_predict = topn_prediction(userid=test_user)
```

```
# Menampilkan hasil Prediksi
topn_predict
```

Hasil yang diperoleh, yaitu:

	itemid	similarity_with_Iu	title	prediction
7	1388	26.624397	Gabbeh (1996)	4.666667
18	359	22.973658	Assignment, The (1997)	4.600000
4	1554	27.364494	Safe Passage (1994)	4.500000
14	1538	24.492453	All Over Me (1997)	4.500000
27	1448	20.846909	My Favorite Season (1993)	4.490052
29	1375	20.627152	Cement Garden, The (1993)	4.333333
26	1466	21.063269	Margaret's Museum (1995)	4.271915
2	1516	31.133267	Wedding Gift, The (1994)	4.000000
23	1467	21.861203	Saint of Fort Washington, The (1993)	4.000000
21	1537	22.061914	Cosi (1996)	4.000000
10	1378	25.787842	Rhyme & Reason (1997)	4.000000
19	1369	22.710078	Forbidden Christ, The (Cristo proibito, Il) (1...	4.000000
3	1550	31.031738	Destiny Turns on the Radio (1995)	3.777778
1	1189	50.362199	Prefontaine (1997)	3.666528
20	1506	22.325504	Nelly & Monsieur Arnaud (1995)	3.610294

19. Evaluasi dengan Mean Absolute Error

```
#Evaluasi dengan Mean Absolute Error
from recsys.preprocessing import train_test_split, get_examples

# get examples as tuples of userids and itemids and labels from normali
ze ratings
raw_examples, raw_labels = get_examples(ratings, labels_column='rating'
)
```

```
# train test split
(x_train, x_test), (y_train, y_test) = train_test_split(examples=raw_ex
amples, labels=raw_labels)

def evaluate(x_test, y_test):
    print('Evaluate the model on {} test data ...'.format(x_test.shape[
0]))
    preds = list(predict(u,i) for (u,i) in x_test)
    mae = np.sum(np.absolute(y_test - np.array(preds))) / x_test.shape[
0]
    print('\nMAE :', mae)
    return mae
```

Hasil yang diperoleh:

```
evaluate(x_test, y_test)

Evaluate the model on 10000 test data ...

MAE : 0.672389703640273
0.672389703640273
```

BAB 7

Pengurangan dimensi

7.1. Definisi Pengurangan Dimensi

Pengurangan dimensi adalah teknik yang digunakan untuk mengurangi jumlah fitur (dimensi) dalam kumpulan data sambil mempertahankan informasi sebanyak mungkin. Ini sering digunakan dalam pembelajaran mesin dan penambangan data untuk memproses data sebelum membangun model, karena data berdimensi tinggi bisa sulit untuk dimodelkan dan juga dapat mengalami kutukan dimensi.

Ada beberapa metode untuk pengurangan dimensi, termasuk:

1. Faktorisasi matriks: Metode ini sering digunakan untuk mengurangi dimensi matriks dan mengekstrak fitur laten dari data. Ini juga dapat digunakan untuk merekonstruksi matriks asli dari matriks yang terdekomposisi, dan membuat prediksi berdasarkan fitur laten.
2. Dekomposisi Nilai Singular (SVD): Metode ini menguraikan matriks data menjadi tiga matriks yang mewakili pentingnya setiap fitur dalam data. Ini sering digunakan untuk mengurangi dimensi matriks jarang yang besar dengan cara yang tidak diawasi.
3. Non-Negative Matrix Factorization (NMF) adalah sebuah teknik pengurangan dimensi yang digunakan untuk memecahkan matriks non-negatif menjadi dua matriks non-negatif yang lebih kecil. NMF menghasilkan fitur yang interpretable dan tidak terkait linier dengan satu sama lain, sehingga lebih mudah dipahami oleh manusia.
4. Explainable Matrix Factorization (EMF) adalah sebuah teknik pengurangan dimensi yang mirip dengan Non-Negative Matrix Factorization (NMF), tetapi memberikan interpretasi yang lebih baik mengenai apa yang dicapai oleh setiap fitur yang dihasilkan. EMF sering digunakan dalam aplikasi seperti analisis topik dan klasifikasi teks, di mana interpretasi yang baik tentang apa yang mewakili setiap fitur sangat penting.

7.2. Implementasi Pengurangan Dimensi

A. Singular Value Decomposition

Hal yang dilakukan adalah:

1. Download Tools

```
import os

if not (os.path.exists("recsys.zip") or os.path.exists("recsys")):
    !wget https://github.com/nzhinusoftcm/review-on-collaborative-filtering/raw/master/recsys.zip
    !unzip recsys.zip
```

Hasil yang diperoleh, yaitu:

```
--2023-01-03 02:44:02--  https://github.com/nzhinusoftcm/review-on-collaborative-filtering/raw/mast
Resolving github.com (github.com)... 20.27.177.113
Connecting to github.com (github.com)|20.27.177.113|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/nzhinusoftcm/review-on-collaborative-filtering/master/r
--2023-01-03 02:44:02--  https://raw.githubusercontent.com/nzhinusoftcm/review-on-collaborative-fil
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.111.133, 185.199.108.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.111.133|:443... connect
HTTP request sent, awaiting response... 200 OK
Length: 15312323 (15M) [application/zip]
Saving to: ‘recsys.zip’

recsys.zip          100%[===================>]  14.60M  --.-KB/s    in 0.1s

2023-01-03 02:44:04 (109 MB/s) - ‘recsys.zip’ saved [15312323/15312323]

Archive:  recsys.zip
   creating: recsys/
  inflating: recsys/datasets.py
  inflating: recsys/preprocessing.py
  inflating: recsys/utils.py
  inflating: recsys/requirements.txt
   creating: recsys/.vscode/
  inflating: recsys/.vscode/settings.json
   creating: recsys/__pycache__/
  inflating: recsys/__pycache__/datasets.cpython-36.pyc
  inflating: recsys/__pycache__/datasets.cpython-37.pyc
  inflating: recsys/__pycache__/utils.cpython-36.pyc
  inflating: recsys/__pycache__/preprocessing.cpython-37.pyc
  inflating: recsys/__pycache__/datasets.cpython-38.pyc
  inflating: recsys/__pycache__/preprocessing.cpython-36.pyc
  inflating: recsys/__pycache__/preprocessing.cpython-38.pyc
   creating: recsys/memories/
  inflating: recsys/memories/ItemToItem.py
  inflating: recsys/memories/UserToUser.py
   creating: recsys/memories/__pycache__/
  inflating: recsys/memories/__pycache__/UserToUser.cpython-36.pyc
  inflating: recsys/memories/__pycache__/UserToUser.cpython-37.pyc
```

2. Import Persyaratan

```
Import Persyaratan
```

```
matplotlib==3.2.2 numpy==1.19.2 pandas==1.0.5 python==3.7 scikit-learn==0.24.1
scikit-surprise==1.1.1 scipy==1.6.2
```

3. Import Library atau Package

```
#Import library atau package
from recsys.datasets import mlLatestSmall, ml100k, ml1m
from sklearn.preprocessing import LabelEncoder
from scipy.sparse import csr_matrix

import pandas as pd
import numpy as np
import os
```

4. Memuat rating movieland

```
#Download rating movielan
ratings, movies = mlLatestSmall.load()
```

Hasil yang diperoleh:

```
Download data 100.5%
Successfully downloaded ml-latest-small.zip 978202 bytes.
Unzipping the ml-latest-small.zip zip file ...
```

5. melihat matriks rating

```
#melihat matriks rating
pd.crosstab(ratings.userid, ratings.itemid, ratings.rating, aggfunc=sum
)
```

hasil yang didapatkan:

itemid	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573	19
userid																
1	4.0	NaN	4.0	NaN	NaN	4.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
5	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
606	2.5	NaN	NaN	NaN	NaN	NaN	2.5	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
607	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
608	2.5	2.0	2.0	NaN	NaN	NaN	NaN	NaN	NaN	4.0	...	NaN	NaN	NaN	NaN	
609	3.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	...	NaN	NaN	NaN	NaN	
610	5.0	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	

610 rows x 9724 columns

6. Menormalkan data degan mengurangi setiap peringkat rata-rata pengguna

```
# get user's mean rating
umean = ratings.groupby(by='userid')['rating'].mean()
```

```
def rating_matrix(ratings):
    """
    1. Fill NaN values with item's average ratings
    2. Normalize ratings by subtracting user's mean ratings

    :param ratings : DataFrame of ratings data
    :return
        - R : Numpy array of normalized ratings
        - df : DataFrame of normalized ratings
    """

    # fill missing values with item's average ratings
    df = pd.crosstab(ratings.userid, ratings.itemid, ratings.rating, ag
gfunc=sum)
    df = df.fillna(df.mean(axis=0))

    # subtract user's mean ratings to normalize data
    df = df.subtract(umean, axis=0)

    # convert our dataframe to numpy array
    R = df.to_numpy()

    return R, df

# generate rating matrix by calling function rating_matrix
R, df = rating_matrix(ratings)
```

7. tampilan matriks peringkat akhir

```
df
```

hasil yang diperoleh:

itemid	1	2	3	4	5	6	7	8	9
userid									
1	-0.366379	-0.934561	-0.366379	-2.009236	-1.294951	-0.366379	-1.181194	-1.491379	-1.241379
2	-0.027346	-0.516458	-0.688660	-1.591133	-0.876847	-0.002197	-0.763091	-1.073276	-0.823276
3	1.485033	0.995921	0.823718	-0.078755	0.635531	1.510181	0.749288	0.439103	0.689103
4	0.365375	-0.123737	-0.295940	-1.198413	-0.484127	0.390523	-0.370370	-0.680556	-0.430556
5	0.363636	-0.204545	-0.376748	-1.279221	-0.564935	0.309715	-0.451178	-0.761364	-0.511364
...	...	...	...	...	...	...	...	...	...
606	-1.157399	-0.225581	-0.397784	-1.300256	-0.585971	0.288679	-1.157399	-0.782399	-0.532399
607	0.213904	-0.354278	-0.526481	-1.428953	-0.714668	0.159982	-0.600911	-0.911096	-0.661096
608	-0.634176	-1.134176	-1.134176	-0.777033	-0.062747	0.811903	0.051009	-0.259176	-0.009176
609	-0.270270	0.161548	-0.010655	-0.913127	-0.198842	0.675808	-0.085085	-0.395270	-0.145270
610	1.311444	-0.256738	-0.428941	-1.331413	-0.617127	1.311444	-0.503371	-0.813556	-0.563556

610 rows × 9724 columns

8. menyandikan pengguna dan id item sehingga nilainya berkisar dari 0 hingga 909 (untuk pengguna) dan dari 0 hingga 9723 (untuk item)

```
#menyandikan pengguna dan id item
users = sorted(ratings['userid'].unique())
items = sorted(ratings['itemid'].unique())

# create our id encoders
uencoder = LabelEncoder()
iencoder = LabelEncoder()

# fit our label encoder
uencoder.fit(users)
iencoder.fit(items)
```

9. menerapkan algoritma SVD

```
#menerapkan algoritma SVD
class SVD:

    def __init__(self, umeam):
        """
        :param
            - umean : mean ratings of users
        """
        self.umean = umean.to_numpy()

        # init svd resultant matrices
        self.P = np.array([])
        self.S = np.array([])
        self.Qh = np.array([])

        # init users and items latent factors
        self.u_factors = np.array([])
        self.i_factors = np.array([])
```

```
    def fit(self, R):
        """
        Fit the SVD model with rating matrix R
        """
        P, s, Qh = np.linalg.svd(R, full_matrices=False)

        self.P = P
        self.S = np.diag(s)
        self.Qh = Qh

        # latent factors of users (u_factors) and items (i_factors)
        self.u_factors = np.dot(self.P, np.sqrt(self.S))
        self.i_factors = np.dot(np.sqrt(self.S), self.Qh)

    def predict(self, userid, itemid):
        """
        Make rating prediction for a given user on an item

        :param
            - userid : user's id
            - itemid : item's id

        :return
            - r_hat : predicted rating
        """
        # encode user and item ids
        u = uencoder.transform([userid])[0]
        i = iencoder.transform([itemid])[0]

        # the predicted rating is the dot product between the uth row
        # of u_factors and the ith column of i_factors
        r_hat = np.dot(self.u_factors[u,:], self.i_factors[:,i])

        # add the mean rating of user u to the predicted value
        r_hat += self.umean[u]

        return r_hat


    def recommend(self, userid):
        """
        :param
            - userid : user's id
        """
        # encode user
        u = uencoder.transform([userid])[0]
```

```
        # the dot product between the uth row of u_factors and i_factor
s returns
        # the predicted value for user u on all items
        predictions = np.dot(self.u_factors[u,:], self.i_factors) + sel
f.umean[u]

        # sort item ids in decreasing order of predictions
        top_idx = np.flip(np.argsort(predictions))

        # decode indices to get their corresponding itemids
        top_items = iencoder.inverse_transform(top_idx)

        # sorted predictions
        preds = predictions[top_idx]

        return top_items, preds
```

10. buat model SVD dan berikan nilai rata-rata pengguna; Sesuaikan model dengan matriks peringkat yang dinormalisasi R.

```
# create our svd model
svd = SVD(umean)

# fit our model with normalized ratings
svd.fit(R)
```

11. membuat beberapa prediksi untuk pengguna menggunakan function predict dari kelas SVD

```
#rating teratas
ratings.head(10)
```

Hasil yag diperoleh, yaitu:

	userid	itemid	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
5	1	70	3.0	964982400
6	1	101	5.0	964980868
7	1	110	4.0	964982176
8	1	151	5.0	964984041
9	1	157	5.0	964984100

12. Melihat prediksi masuk akal atau tidak

```
# user for which we make predictions
userid = 1

# list of items for which we are making predictions for user 1
items = [1,3,6,47,50,70,101,110,151,157]

# predictions
for itemid in items:
    r = svd.predict(userid=userid, itemid=itemid)
    print('prediction for userid={} and itemid={} : {}'.format(userid,
itemid, r))
```

Hasil yang diperoleh, yaitu:

```
prediction for userid=1 and itemid=1 : 3.9999999999999996
prediction for userid=1 and itemid=3 : 4.000000000000036
prediction for userid=1 and itemid=6 : 3.9999999999999867
prediction for userid=1 and itemid=47 : 5.0
prediction for userid=1 and itemid=50 : 4.999999999999964
prediction for userid=1 and itemid=70 : 2.999999999999981
prediction for userid=1 and itemid=101 : 5.000000000000006
prediction for userid=1 and itemid=110 : 3.9999999999999862
prediction for userid=1 and itemid=151 : 5.000000000000115
prediction for userid=1 and itemid=157 : 5.000000000000028
```

13. Membuat rekomendasi

```
userid = 1

# items sorted in decreasing order of predictions for user 1
sorted_items, preds = svd.recommend(userid=userid)

##
```

```
# Now let's exclud from that sorted list items already purchased by the
 user
##

# list of items rated by the user
uitems = ratings.loc[ratings.userid == userid].itemid.to_list()

# remove from sorted_items items already in uitems and pick the top 30
ones
# as recommendation list
top30 = np.setdiff1d(sorted_items, uitems, assume_unique=True)[:30]

# get corresponding predictions from the top30 items
top30_idx = list(np.where(sorted_items == idx)[0][0] for idx in top30)
top30_predictions = preds[top30_idx]

# find corresponding movie titles
zipped_top30 = list(zip(top30,top30_predictions))
top30 = pd.DataFrame(zipped_top30, columns=['itemid','predictions'])
List = pd.merge(top30, movies, on='itemid', how='inner')

# show the list
List
```

Hasil yang diperoleh:

	itemid	predictions	title	genres
0	148	5.0	Awfully Big Adventure, An (1995)	Drama
1	6086	5.0	I, the Jury (1982)	Crime|Drama|Thriller
2	136445	5.0	George Carlin: Back in Town (1996)	Comedy
3	6201	5.0	Lady Jane (1986)	Drama|Romance
4	2075	5.0	Mephisto (1981)	Drama|War
5	6192	5.0	Open Hearts (Elsker dig for evigt) (2002)	Romance
6	117531	5.0	Watermark (2014)	Documentary
7	158398	5.0	World of Glory (1991)	Comedy
8	6021	5.0	American Friend, The (Amerikanische Freund, De...	Crime|Drama|Mystery|Thriller
9	136556	5.0	Kung Fu Panda: Secrets of the Masters (2011)	Animation|Children
10	136447	5.0	George Carlin: You Are All Diseased (1999)	Comedy
11	136503	5.0	Tom and Jerry: Shiver Me Whiskers (2006)	Animation|Children|Comedy
12	134095	5.0	My Love (2006)	Animation|Drama
13	3851	5.0	I'm the One That I Want (2000)	Comedy
14	136469	5.0	Larry David: Curb Your Enthusiasm (1999)	Comedy
15	158882	5.0	All Yours (2016)	Comedy|Drama|Romance
16	134004	5.0	What Love Is (2007)	Comedy|Romance
17	67618	5.0	Strictly Sexual (2008)	Comedy|Drama|Romance
18	3567	5.0	Bossa Nova (2000)	Comedy|Drama|Romance
19	158027	5.0	SORI: Voice from the Heart (2016)	Drama|Sci-Fi
20	59814	5.0	Ex Drummer (2007)	Comedy|Crime|Drama|Horror

Kesimpulan:

Untuk meningkatkan pemfilteran kolaboratif berbasis memori SVD dapat diterapkan karena SVD digunakan untuk meningkatkan pemfilteran kolaboratif berbasis pengguna dan item. Alih-alih menghitung kesamaan antara peringkat pengguna atau item, kita dapat merepresentasikan pengguna dan item dengan faktor laten terkait yang diekstrak dari algoritme SVD.

B. Matrix Factorization

Hal yang dilakukan adalah:

1. Download Tools

```
import os

if not (os.path.exists("recsys.zip") or os.path.exists("recsys")):
    !wget https://github.com/nzhinusoftcm/review-on-collaborative-filtering/raw/master/recsys.zip
    !unzip recsys.zip
```

Hasil yang diperoleh, yaitu:

```
--2023-01-03 02:44:02--  https://github.com/nzhinusoftcm/review-on-collaborative-filtering/raw/mast
Resolving github.com (github.com)... 20.27.177.113
Connecting to github.com (github.com)|20.27.177.113|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/nzhinusoftcm/review-on-collaborative-filtering/master/r
--2023-01-03 02:44:02--  https://raw.githubusercontent.com/nzhinusoftcm/review-on-collaborative-fil
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.111.133, 185.199.108.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.111.133|:443... connect
HTTP request sent, awaiting response... 200 OK
Length: 15312323 (15M) [application/zip]
Saving to: ‘recsys.zip’

recsys.zip          100%[===================>]  14.60M  --.-KB/s    in 0.1s

2023-01-03 02:44:04 (109 MB/s) - ‘recsys.zip’ saved [15312323/15312323]

Archive:  recsys.zip
   creating: recsys/
  inflating: recsys/datasets.py
  inflating: recsys/preprocessing.py
  inflating: recsys/utils.py
  inflating: recsys/requirements.txt
   creating: recsys/.vscode/
  inflating: recsys/.vscode/settings.json
   creating: recsys/__pycache__/
  inflating: recsys/__pycache__/datasets.cpython-36.pyc
  inflating: recsys/__pycache__/datasets.cpython-37.pyc
  inflating: recsys/__pycache__/utils.cpython-36.pyc
  inflating: recsys/__pycache__/preprocessing.cpython-37.pyc
  inflating: recsys/__pycache__/datasets.cpython-38.pyc
  inflating: recsys/__pycache__/preprocessing.cpython-36.pyc
  inflating: recsys/__pycache__/preprocessing.cpython-38.pyc
   creating: recsys/memories/
  inflating: recsys/memories/ItemToItem.py
  inflating: recsys/memories/UserToUser.py
   creating: recsys/memories/__pycache__/
  inflating: recsys/memories/__pycache__/UserToUser.cpython-36.pyc
  inflating: recsys/memories/__pycache__/UserToUser.cpython-37.pyc
```

2. Import Persyaratan

```
Import Persyaratan

matplotlib==3.2.2 numpy==1.19.2 pandas==1.0.5 python==3.7 scikit-learn==0.24.1 scikit-surprise==1.1.1 scipy==1.6.2
```

3. Import Library atau Package

```
#Import Library atau package
from recsys.preprocessing import mean_ratings
from recsys.preprocessing import normalized_ratings
from recsys.preprocessing import ids_encoder
from recsys.preprocessing import train_test_split
from recsys.preprocessing import rating_matrix
from recsys.preprocessing import get_examples
from recsys.preprocessing import scale_ratings

from recsys.datasets import ml100k
from recsys.datasets import ml1m

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

import os
```

4. Pendefinisian Model

```
#Pendefinisian Model
class MatrixFactorization:

    def __init__(self, m, n, k=10, alpha=0.001, lamb=0.01):
        """
        Initialization of the model
        : param
            - m : number of users
            - n : number of items
            - k : length of latent factor, both for users and items. 50
 by default
            - alpha : learning rate. 0.001 by default
            - lamb : regularizer parameter. 0.02 by default
        """
        np.random.seed(32)

        # initialize the latent factor matrices P and Q (of shapes (m,k
) and (n,k) respectively) that will be learnt
        self.k = k
        self.P = np.random.normal(size=(m, k))
        self.Q = np.random.normal(size=(n, k))

        # hyperparameter initialization
```

```
        self.alpha = alpha
        self.lamb = lamb

        # training history
        self.history = {
            "epochs":[],
            "loss":[],
            "val_loss":[],
            "lr":[]
        }

    def print_training_parameters(self):
        print('Training Matrix Factorization Model ...')
        print(f'k={self.k} \t alpha={self.alpha} \t lambda={self.lamb}'
)

    def update_rule(self, u, i, error):
        self.P[u] = self.P[u] + self.alpha * (error * self.Q[i] - self.
lamb * self.P[u])
        self.Q[i] = self.Q[i] + self.alpha * (error * self.P[u] - self.
lamb * self.Q[i])

    def mae(self,  x_train, y_train):
        """
        returns the Mean Absolute Error
        """
        # number of training exemples
        M = x_train.shape[0]
        error = 0
        for pair, r in zip(x_train, y_train):
            u, i = pair
            error += abs(r - np.dot(self.P[u], self.Q[i]))
        return error/M

    def print_training_progress(self, epoch, epochs, error, val_error,
steps=5):
        if epoch == 1 or epoch % steps == 0 :
                print("epoch {}/{} - loss : {} - val_loss : {}".format(
epoch, epochs, round(error,3), round(val_error,3)))

    def learning_rate_schedule(self, epoch, target_epochs = 20):
        if (epoch >= target_epochs) and (epoch % target_epochs == 0):
                factor = epoch // target_epochs
                self.alpha = self.alpha * (1 / (factor * 20))
                print("\nLearning Rate : {}\n".format(self.alpha))

    def fit(self, x_train, y_train, validation_data, epochs=1000):
        """
```

```
        Train latent factors P and Q according to the training set

        :param
            - x_train : training pairs (u,i) for which rating r_ui is k
nown
            - y_train : set of ratings r_ui for all training pairs (u,i
)
            - validation_data : tuple (x_test, y_test)
            - epochs : number of time to loop over the entire training
set.
            1000 epochs by default

        Note that u and i are encoded values of userid and itemid
        """
        self.print_training_parameters()

        # validation data
        x_test, y_test = validation_data

        # loop over the number of epochs
        for epoch in range(1, epochs+1):

            # for each pair (u,i) and the corresponding rating r
            for pair, r in zip(x_train, y_train):

                # get encoded values of userid and itemid from pair
                u,i = pair

                # compute the predicted rating r_hat
                r_hat = np.dot(self.P[u], self.Q[i])

                # compute the prediction error
                e = abs(r - r_hat)

                # update rules
                self.update_rule(u, i, e)

            # training and validation error  after this epochs
            error = self.mae(x_train, y_train)
            val_error = self.mae(x_test, y_test)

            # update history
            self.history['epochs'].append(epoch)
            self.history['loss'].append(error)
            self.history['val_loss'].append(val_error)

            # update history
            self.update_history(epoch, error, val_error)
```

```
            # print training progress after each steps epochs
            self.print_training_progress(epoch, epochs, error, val_erro
r, steps=1)

            # leaning rate scheduler : redure the learning rate as we g
o deeper in the number of epochs
            # self.learning_rate_schedule(epoch)

        return self.history

    def update_history(self, epoch, error, val_error):
        self.history['epochs'].append(epoch)
        self.history['loss'].append(error)
        self.history['val_loss'].append(val_error)
        self.history['lr'].append(self.alpha)

    def evaluate(self, x_test, y_test):
        """
        compute the global error on the test set
        :param x_test : test pairs (u,i) for which rating r_ui is known
        :param y_test : set of ratings r_ui for all test pairs (u,i)
        """
        error = self.mae(x_test, y_test)
        print(f"validation error : {round(error,3)}")

        return error

    def predict(self, userid, itemid):
        """
        Make rating prediction for a user on an item
        :param userid
        :param itemid
        :return r : predicted rating
        """
        # encode user and item ids to be able to access their latent fa
ctors in
        # matrices P and Q
        u = uencoder.transform([userid])[0]
        i = iencoder.transform([itemid])[0]

        # rating prediction using encoded ids. Dot product between P_u
and Q_i
        r = np.dot(self.P[u], self.Q[i])
        return r

    def recommend(self, userid, N=30):
        """
```

```
        make to N recommendations for a given user

        :return(top_items,preds) : top N items with the highest predict
ions
        with their corresponding predictions
        """
        # encode the userid
        u = uencoder.transform([userid])[0]

        # predictions for users userid on all product
        predictions = np.dot(self.P[u], self.Q.T)

        # get the indices of the top N predictions
        top_idx = np.flip(np.argsort(predictions))[:N]

        # decode indices to get their corresponding itemids
        top_items = iencoder.inverse_transform(top_idx)

        # take corresponding predictions for top N indices
        preds = predictions[top_idx]

        return top_items, preds
```

```
epochs = 10
```

5. Evaluasi rating mentah

```
# load the ml100k dataset
ratings, movies = ml100k.load()

ratings, uencoder, iencoder = ids_encoder(ratings)

m = ratings.userid.nunique()    # total number of users
n = ratings.itemid.nunique()    # total number of items

# get examples as tuples of userids and itemids and labels from normali
ze ratings
raw_examples, raw_labels = get_examples(ratings)

# train test split
(x_train, x_test), (y_train, y_test) = train_test_split(examples=raw_ex
amples, labels=raw_labels)
```

Hasil yang diperoleh:

```
Download data 100.2%
Successfully downloaded ml-100k.zip 4924029 bytes.
Unzipping the ml-100k.zip zip file ...
```

```
# create the model
MF = MatrixFactorization(m, n, k=10, alpha=0.01, lamb=1.5)

# fit the model on the training set
history = MF.fit(x_train, y_train, epochs=epochs, validation_data=(x_te
st, y_test))
```

Hasil yang diperoleh:

```
Training Matrix Factorization Model ...
k=10     alpha=0.01      lambda=1.5
epoch 1/10 - loss : 2.734 - val_loss : 2.779
epoch 2/10 - loss : 1.764 - val_loss : 1.794
epoch 3/10 - loss : 1.592 - val_loss : 1.614
epoch 4/10 - loss : 1.538 - val_loss : 1.556
epoch 5/10 - loss : 1.515 - val_loss : 1.531
epoch 6/10 - loss : 1.503 - val_loss : 1.517
epoch 7/10 - loss : 1.496 - val_loss : 1.509
epoch 8/10 - loss : 1.491 - val_loss : 1.504
epoch 9/10 - loss : 1.488 - val_loss : 1.5
epoch 10/10 - loss : 1.486 - val_loss : 1.497
```

```
MF.evaluate(x_test, y_test)
```

Hasil yang didapatkan:

```
validation error : 1.497
1.4973507972141993
```

6. Evaluasi pada Rating yang dinormalisasi

```
# load data
ratings, movies = ml100k.load()

ratings, uencoder, iencoder = ids_encoder(ratings)

m = ratings['userid'].nunique()    # total number of users
n = ratings['itemid'].nunique()    # total number of items

# normalize ratings by substracting means
normalized_column_name = "norm_rating"
ratings = normalized_ratings(ratings, norm_column=normalized_column_nam
e)

# get examples as tuples of userids and itemids and labels from normali
ze ratings
raw_examples, raw_labels = get_examples(ratings, labels_column=normaliz
ed_column_name)

# train test split
(x_train, x_test), (y_train, y_test) = train_test_split(examples=raw_ex
amples, labels=raw_labels)
```

```
# create the model
MF = MatrixFactorization(m, n, k=10, alpha=0.01, lamb=1.5)

# fit the model on the training set
history = MF.fit(x_train, y_train, epochs=epochs, validation_data=(x_te
st, y_test))
```

hasil yang diperoleh:

```
Training Matrix Factorization Model ...
k=10     alpha=0.01      lambda=1.5
epoch 1/10 - loss : 0.851 - val_loss : 0.847
epoch 2/10 - loss : 0.831 - val_loss : 0.831
epoch 3/10 - loss : 0.828 - val_loss : 0.829
epoch 4/10 - loss : 0.827 - val_loss : 0.828
epoch 5/10 - loss : 0.827 - val_loss : 0.828
epoch 6/10 - loss : 0.826 - val_loss : 0.828
epoch 7/10 - loss : 0.826 - val_loss : 0.828
epoch 8/10 - loss : 0.826 - val_loss : 0.828
epoch 9/10 - loss : 0.826 - val_loss : 0.828
epoch 10/10 - loss : 0.826 - val_loss : 0.828
```

9. Prediksi

```
ratings.userid = uencoder.inverse_transform(ratings.userid.to_list())
ratings.itemid = uencoder.inverse_transform(ratings.itemid.to_list())
ratings.head(5)
```

Hasil yang diperoleh:

	userid	itemid	rating	rating_mean	norm_rating
0	1	1	5	4.188679	0.811321
1	1	48	5	4.188679	0.811321
2	1	145	5	4.188679	0.811321
3	1	254	4	4.188679	-0.188679
4	1	514	5	4.188679	0.811321

```
4.188679 + MF.predict(userid=1, itemid=1) # add the mean because we hav
e used the normalised rating
```

Hasil yang diperoleh: `4.188679163563357`

Kesimpulan:

1. Dengan model Faktorisasi Matriks, P dan Q faktor laten tidak dapat ditafsirkan karena mengandung nilai negatif atau positif yang berubah-ubah. Ini membuat model Faktorisasi Matriks menjadi tidak dapat dijelaskan.
2. Algoritma Faktorisasi Matriks Non-negatif (NMF) adalah varian dari Faktorisasi Matriks yang menghasilkan faktor laten yang dapat dijelaskan untuk dan dengan membatasi nilainya dalam kisaran [0,1]. Ini memungkinkan interpretasi probabilitas dari faktor-faktor laten ini, sehingga dapat dijelaskan.

C. Non-negative Matrix Factorization

Hal yang dilakukan, yaitu:

1. Download Tools

```
import os

if not (os.path.exists("recsys.zip") or os.path.exists("recsys")):
    !wget https://github.com/nzhinusoftcm/review-on-collaborative-
filtering/raw/master/recsys.zip
    !unzip recsys.zip
```

2. Import Packages

```
#Import Packages
from recsys.preprocessing import mean_ratings
from recsys.preprocessing import normalized_ratings
from recsys.preprocessing import ids_encoder
from recsys.preprocessing import train_test_split
from recsys.preprocessing import rating_matrix
from recsys.preprocessing import get_examples
from recsys.preprocessing import scale_ratings

from recsys.datasets import ml1m
from recsys.datasets import ml100k

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

import os
```

3. Load dan Proses data

```
# load data
ratings, movies = ml100k.load()

# prepare data
ratings, uencoder, iencoder = ids_encoder(ratings)

# convert ratings from dataframe to numpy array
np_ratings = ratings.to_numpy()

# get examples as tuples of userids and itemids and labels from normali
ze ratings
```

```
raw_examples, raw_labels = get_examples(ratings, labels_column="rating"
)

# train test split
(x_train, x_test), (y_train, y_test) = train_test_split(examples=raw_ex
amples, labels=raw_labels)
```

4. Membangun Model Non-negative Matrix Factorization

```
class NMF:

    def __init__(self, ratings, m, n, uencoder, iencoder, K=10, lambda_
P=0.01, lambda_Q=0.01):

        np.random.seed(32)

        # initialize the latent factor matrices P and Q (of shapes (m,k
) and (n,k) respectively) that will be learnt
        self.ratings = ratings
        self.np_ratings = ratings.to_numpy()
        self.K = K
        self.P = np.random.rand(m, K)
        self.Q = np.random.rand(n, K)

        # hyper parameter initialization
        self.lambda_P = lambda_P
        self.lambda_Q = lambda_Q

        # initialize encoders
        self.uencoder = uencoder
        self.iencoder = iencoder

        # training history
        self.history = {
            "epochs": [],
            "loss": [],
            "val_loss": [],
        }

    def print_training_parameters(self):
        print('Training NMF ...')
        print(f'k={self.K}')

    def mae(self, x_train, y_train):
        """
        returns the Mean Absolute Error
        """
```

```
        # number of training examples
        m = x_train.shape[0]
        error = 0
        for pair, r in zip(x_train, y_train):
            u, i = pair
            error += abs(r - np.dot(self.P[u], self.Q[i]))
        return error / m

    def update_rule(self, u, i, error):
        I = self.np_ratings[self.np_ratings[:, 0] == u][:, [1, 2]]
        U = self.np_ratings[self.np_ratings[:, 1] == i][:, [0, 2]]

        num = self.P[u] * np.dot(self.Q[I[:, 0]].T, I[:, 1])
        dem = np.dot(self.Q[I[:, 0]].T, np.dot(self.P[u], self.Q[I[:, 0
]].T)) + self.lambda_P * len(I) * self.P[u]
        self.P[u] = num / dem

        num = self.Q[i] * np.dot(self.P[U[:, 0]].T, U[:, 1])
        dem = np.dot(self.P[U[:, 0]].T, np.dot(self.P[U[:, 0]], self.Q[
i].T)) + self.lambda_Q * len(U) * self.Q[i]
        self.Q[i] = num / dem

    @staticmethod
    def print_training_progress(epoch, epochs, error, val_error, steps=
5):
        if epoch == 1 or epoch % steps == 0:
            print(f"epoch {epoch}/{epochs} - loss : {round(error, 3)} -
 val_loss : {round(val_error, 3)}")

    def fit(self, x_train, y_train, validation_data, epochs=10):

        self.print_training_parameters()
        x_test, y_test = validation_data
        for epoch in range(1, epochs+1):
            for pair, r in zip(x_train, y_train):
                u, i = pair
                r_hat = np.dot(self.P[u], self.Q[i])
                e = abs(r - r_hat)
                self.update_rule(u, i, e)
            # training and validation error  after this epochs
            error = self.mae(x_train, y_train)
            val_error = self.mae(x_test, y_test)
            self.update_history(epoch, error, val_error)
            self.print_training_progress(epoch, epochs, error, val_erro
r, steps=1)

        return self.history
```

```
    def update_history(self, epoch, error, val_error):
        self.history['epochs'].append(epoch)
        self.history['loss'].append(error)
        self.history['val_loss'].append(val_error)

    def evaluate(self, x_test, y_test):
        error = self.mae(x_test, y_test)
        print(f"validation error : {round(error,3)}")
        print('MAE : ', error)
        return error

    def predict(self, userid, itemid):
        u = self.uencoder.transform([userid])[0]
        i = self.iencoder.transform([itemid])[0]
        r = np.dot(self.P[u], self.Q[i])
        return r

    def recommend(self, userid, N=30):
        # encode the userid
        u = self.uencoder.transform([userid])[0]

        # predictions for users userid on all product
        predictions = np.dot(self.P[u], self.Q.T)

        # get the indices of the top N predictions
        top_idx = np.flip(np.argsort(predictions))[:N]

        # decode indices to get their corresponding itemids
        top_items = self.iencoder.inverse_transform(top_idx)

        # take corresponding predictions for top N indices
        preds = predictions[top_idx]

        return top_items, preds
```

5. Membuat model training

```
m = ratings['userid'].nunique()    # total number of users
n = ratings['itemid'].nunique()    # total number of items

# create and train the model
nmf = NMF(ratings, m, n, uencoder, iencoder, K=10, lambda_P=0.6, lambda
_Q=0.6)
history = nmf.fit(x_train, y_train, epochs=10, validation_data=(x_test,
 y_test))
```

6. Hitung validasi dan MAE

```
nmf.evaluate(x_test, y_test)
```

7. Evaluasi NMF

```
!pip install surprise
import surprise
from surprise import NMF
from surprise import Dataset
from surprise.model_selection import cross_validate

# Load the movielens-100k dataset (download it if needed).
data = Dataset.load_builtin('ml-100k')

# Use the NMF algorithm.
nmf = NMF(n_factors=10, n_epochs=10)

# Run 5-fold cross-validation and print results.
history = cross_validate(nmf, data, measures=['MAE'], cv=5, verbose=Tru
e)
```

Kesimpulan:

NMF memperkenalkan penjelasan untuk MF dengan membatasi dan nilai dalam [1,0]. Ini dapat dianggap sebagai model yang dapat dijelaskan dari dalam. Dimungkinkan untuk menyuntikkan informasi eksternal ke dalam model untuk memberikan penjelasan. Inilah yang dilakukan oleh algoritma Explainable Matrix Factorization (EMF). Itu menghitung skor yang dapat dijelaskan dari kesamaan pengguna atau item yang diambil dari CF berbasis pengguna atau berbasis item.

D. Explainable Matrix Factorization (EMF)

Hal yang dilakukan:

1. Download Tools

```
import os

if not (os.path.exists("recsys.zip") or os.path.exists("recsys")):
    !wget https://github.com/nzhinusoftcm/review-on-collaborative-
filtering/raw/master/recsys.zip
    !unzip recsys.zip
```

2. Import Packages

```
#Import packages
from recsys.memories.UserToUser import UserToUser

from recsys.preprocessing import mean_ratings
from recsys.preprocessing import normalized_ratings
from recsys.preprocessing import ids_encoder
from recsys.preprocessing import train_test_split
from recsys.preprocessing import rating_matrix
from recsys.preprocessing import get_examples

from recsys.datasets import ml100k, ml1m

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

import sys
import os
```

3. Compute Explainable Score

```
#Compute Explainable Scores
def explainable_score(user2user, users, items, theta=0):

    def _progress(count):
        sys.stdout.write('\rCompute Explainable score. Progress status
: %.1f%%'%(float(count/len(users))*100.0))
        sys.stdout.flush()
    # initialize explainable score to zeros
    W = np.zeros((len(users), len(items)))

    for count, u in enumerate(users):
        candidate_items = user2user.find_user_candidate_items(u)

        for i in candidate_items:
            user_who_rated_i, similar_user_who_rated_i = \
                user2user.similar_users_who_rated_this_item(u, i)
            if user_who_rated_i.shape[0] == 0:
                w = 0.0
            else:
                w = similar_user_who_rated_i.shape[0] / user_who_rated_
i.shape[0]
            W[u,i] =  w  if w > theta else 0.0
        _progress(count)
```

```
    return W
```

4. Explainable Matrix Factorization Model

```
#Explainable Matrix Factorization Model
class ExplainableMatrixFactorization:

    def __init__(self, m, n, W, alpha=0.001, beta=0.01, lamb=0.1, k=10)
:
        """
            - R : Rating matrix of shape (m,n)
            - W : Explainability Weights of shape (m,n)
            - k : number of latent factors
            - beta : L2 regularization parameter
            - lamb : explainability regularization coefficient
            - theta : threshold above which an item is explainable for
a user
        """
        self.W = W
        self.m = m
        self.n = n

        np.random.seed(64)

        # initialize the latent factor matrices P and Q (of shapes (m,k
) and (n,k) respectively) that will be learnt
        self.k = k
        self.P = np.random.normal(size=(self.m,k))
        self.Q = np.random.normal(size=(self.n,k))

        # hyperparameter initialization
        self.alpha = alpha
        self.beta = beta
        self.lamb = lamb

        # training history
        self.history = {
            "epochs":[],
            "loss":[],
            "val_loss":[],
        }

    def print_training_parameters(self):
        print('Training EMF')
        print(f'k={self.k} \t alpha={self.alpha} \t beta={self.beta} \t
 lambda={self.lamb}')
```

```
    def update_rule(self, u, i, error):
        self.P[u] = self.P[u] + \
            self.alpha*(2 * error*self.Q[i] - self.beta*self.P[u] - sel
f.lamb*(self.P[u] - self.Q[i]) * self.W[u,i])

        self.Q[i] = self.Q[i] + \
            self.alpha*(2 * error*self.P[u] - self.beta*self.Q[i] + sel
f.lamb*(self.P[u] - self.Q[i]) * self.W[u,i])

    def mae(self,  x_train, y_train):
        """
        returns the Mean Absolute Error
        """
        # number of training exemples
        M = x_train.shape[0]
        error = 0
        for pair, r in zip(x_train, y_train):
            u, i = pair
            error += np.absolute(r - np.dot(self.P[u], self.Q[i]))
        return error/M

    def print_training_progress(self, epoch, epochs, error, val_error,
steps=5):
        if epoch == 1 or epoch % steps == 0 :
                print(f"epoch {epoch}/{epochs} - loss : {round(error,3)
} - val_loss : {round(val_error,3)}")

    def learning_rate_schedule(self, epoch, target_epochs = 20):
        if (epoch >= target_epochs) and (epoch % target_epochs == 0):
                factor = epoch // target_epochs
                self.alpha = self.alpha * (1 / (factor * 20))
                print("\nLearning Rate : {}\n".format(self.alpha))

    def fit(self, x_train, y_train, validation_data, epochs=10):
        """
        Train latent factors P and Q according to the training set

        :param
            - x_train : training pairs (u,i) for which rating r_ui is k
nown
            - y_train : set of ratings r_ui for all training pairs (u,i
)
            - validation_data : tuple (x_test, y_test)
            - epochs : number of time to loop over the entire training
set.
            10 epochs by default

        Note that u and i are encoded values of userid and itemid
```

```
        """
        self.print_training_parameters()

        # get validation data
        x_test, y_test = validation_data

        for epoch in range(1, epochs+1):
            for pair, r in zip(x_train, y_train):
                u,i = pair
                r_hat = np.dot(self.P[u], self.Q[i])
                e = r - r_hat
                self.update_rule(u, i, error=e)

            # training and validation error  after this epochs
            error = self.mae(x_train, y_train)
            val_error = self.mae(x_test, y_test)
            self.update_history(epoch, error, val_error)
            self.print_training_progress(epoch, epochs, error, val_erro
r, steps=1)

        return self.history

    def update_history(self, epoch, error, val_error):
        self.history['epochs'].append(epoch)
        self.history['loss'].append(error)
        self.history['val_loss'].append(val_error)

    def evaluate(self, x_test, y_test):
        """
        compute the global error on the test set

        :param
            - x_test : test pairs (u,i) for which rating r_ui is known
            - y_test : set of ratings r_ui for all test pairs (u,i)
        """
        error = self.mae(x_test, y_test)
        print(f"validation error : {round(error,3)}")

    def predict(self, userid, itemid):
        """
        Make rating prediction for a user on an item

        :param
        - userid
        - itemid

        :return
        - r : predicted rating
```

```
        """
        # encode user and item ids to be able to access their latent fa
ctors in
        # matrices P and Q
        u = uencoder.transform([userid])[0]
        i = iencoder.transform([itemid])[0]

        # rating prediction using encoded ids. Dot product between P_u
and Q_i
        r = np.dot(self.P[u], self.Q[i])

        return r

    def recommend(self, userid, N=30):
        """
        make to N recommendations for a given user

        :return
        - (top_items,preds) : top N items with the highest predictions
        """
        # encode the userid
        u = uencoder.transform([userid])[0]

        # predictions for this user on all product
        predictions = np.dot(self.P[u], self.Q.T)

        # get the indices of the top N predictions
        top_idx = np.flip(np.argsort(predictions))[:N]

        # decode indices to get their corresponding itemids
        top_items = iencoder.inverse_transform(top_idx)

        # take corresponding predictions for top N indices
        preds = predictions[top_idx]

        return top_items, preds
```

```
epochs = 10
```

5. Model Evaluation

```
# load data
ratings, movies = ml100k.load()

# encode users and items ids
ratings, uencoder, iencoder = ids_encoder(ratings)
```

```
users = sorted(ratings.userid.unique())
items = sorted(ratings.itemid.unique())

m = len(users)
n = len(items)

# get examples as tuples of userids and itemids and labels from normali
ze ratings
raw_examples, raw_labels = get_examples(ratings)

# train test split
(x_train, x_test), (y_train, y_test) = train_test_split(examples=raw_ex
amples, labels=raw_labels)
```

```
# create the user to user model for similarity measure
usertouser = UserToUser(ratings, movies)

# compute explainable score
W = explainable_score(usertouser, users, items)
```

```
# initialize the model
EMF = ExplainableMatrixFactorization(m, n, W, alpha=0.01, beta=0.4, lam
b=0.01, k=10)

history = EMF.fit(x_train, y_train, epochs=epochs, validation_data=(x_t
est, y_test))
```

```
EMF.evaluate(x_test, y_test)
```

6. Evaluation on normalized data

```
# load data
ratings, movies = ml100k.load()

# encode users and items ids
ratings, uencoder, iencoder = ids_encoder(ratings)

users = sorted(ratings.userid.unique())
items = sorted(ratings.itemid.unique())

m = len(users)
n = len(items)

# normalize ratings by substracting means
```

```
normalized_column_name = "norm_rating"
ratings = normalized_ratings(ratings, norm_column=normalized_column_nam
e)

# get examples as tuples of userids and itemids and labels from normali
ze ratings
raw_examples, raw_labels = get_examples(ratings, labels_column=normaliz
ed_column_name)

# train test split
(x_train, x_test), (y_train, y_test) = train_test_split(examples=raw_ex
amples, labels=raw_labels)
```

```
# initialize the model
EMF = ExplainableMatrixFactorization(m, n, W, alpha=0.022, beta=0.65, l
amb=0.01, k=10)

history = EMF.fit(x_train, y_train, epochs=epochs, validation_data=(x_t
est, y_test))
```

7. Ratings prediction

```
# get list of top N items with their corresponding predicted ratings
userid = 42
recommended_items, predictions = EMF.recommend(userid=userid)

# find corresponding movie titles
top_N = list(zip(recommended_items,predictions))
top_N = pd.DataFrame(top_N, columns=['itemid','predictions'])
top_N.predictions = top_N.predictions + ratings.loc[ratings.userid==use
rid].rating_mean.values[0]
List = pd.merge(top_N, movies, on='itemid', how='inner')

# show the list
List
```

BAB 8

Perbandingan Kinerja

8.1. Definisi Perbandingan Kinerja

Perbandingan kinerja mengacu pada proses mengevaluasi dan membandingkan kinerja dua entitas atau lebih, seperti organisasi, sistem, program, proyek, proses, atau individu. Perbandingan kinerja dapat digunakan untuk mengidentifikasi kekuatan dan kelemahan, menetapkan tolok ukur, dan mendorong peningkatan berkelanjutan.

Ada banyak cara untuk membandingkan kinerja, termasuk:

1. Ukuran keuangan: Bandingkan indikator kinerja keuangan seperti pendapatan, laba, laba atas investasi (ROI), dan efektivitas biaya.
2. Ukuran pelanggan: Bandingkan skor kepuasan pelanggan, loyalitas, tingkat retensi, dan indikator kepuasan dan loyalitas pelanggan lainnya.
3. Pengukuran proses internal: Bandingkan efisiensi proses, akurasi, kecepatan, dan indikator kinerja proses internal lainnya.
4. Pengukuran pembelajaran dan pertumbuhan: Bandingkan program pelatihan dan pengembangan, tingkat keterlibatan dan retensi karyawan, serta indikator pembelajaran dan pertumbuhan lainnya.

Penting untuk mempertimbangkan konteks dan batasan perbandingan saat mengevaluasi kinerja. Faktor-faktor seperti ukuran, industri, lokasi, dan variabel lainnya dapat memengaruhi kinerja dan harus diperhitungkan.

8.2. Perbandingan kinerja

Hal yang dilakukan adalah:

1. Download Tools

```
#Download Tools
import os

if not (os.path.exists("recsys.zip") or os.path.exists("recsys")):
    !wget https://github.com/nzhinusoftcm/review-on-collaborative-
filtering/raw/master/recsys.zip
    !unzip recsys.zip
```

2. Import Packages

```
#Import Packages
from recsys.memories.UserToUser import UserToUser
from recsys.memories.ItemToItem import ItemToItem

from recsys.models.MatrixFactorization import MF
from recsys.models.ExplainableMF import EMF, explainable_score

from recsys.preprocessing import normalized_ratings
from recsys.preprocessing import train_test_split
from recsys.preprocessing import rating_matrix
from recsys.preprocessing import scale_ratings
from recsys.preprocessing import mean_ratings
from recsys.preprocessing import get_examples
from recsys.preprocessing import ids_encoder

from recsys.datasets import ml100k
from recsys.datasets import ml1m

from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

import os
```

3. Results on Movielens 100K
   a. User-based CF

```
# load data
ratings, movies = ml100k.load()

# prepare data
ratings, uencoder, iencoder = ids_encoder(ratings)

# get examples as tuples of userids and itemids and labels from n
ormalize ratings
raw_examples, raw_labels = get_examples(ratings, labels_column='r
ating')

# train test split
(x_train, x_test), (y_train, y_test) = train_test_split(examples=
raw_examples, labels=raw_labels)

# evaluate with Euclidean distance
```

```
usertouser = UserToUser(ratings, movies, metric='euclidean')
print("==========================")
usertouser.evaluate(x_test, y_test)
```

Hasil yang diperoleh:

```
Normalize users ratings ...
Initialize the similarity model ...
Compute nearest neighbors ...
User to user recommendation model created with success ...
==========================
Evaluate the model on 10000 test data ...

MAE : 0.8125945111976461
0.8125945111976461
```

```
# evaluate with cosine similarity

usertouser = UserToUser(ratings, movies, metric='cosine')
print("==========================")
usertouser.evaluate(x_test, y_test)
```

Hasil yang diperoleh:

```
Normalize users ratings ...
Initialize the similarity model ...
Compute nearest neighbors ...
User to user recommendation model created with success ...
========================
Evaluate the model on 10000 test data ...

MAE : 0.7505910931068639
0.7505910931068639
```

b. Item-based CF

```
# load data
ratings, movies = ml100k.load()

# prepare data
ratings, uencoder, iencoder = ids_encoder(ratings)

# get examples as tuples of userids and itemids and labels from n
ormalize ratings
raw_examples, raw_labels = get_examples(ratings, labels_column='r
ating')

# train test split
(x_train, x_test), (y_train, y_test) = train_test_split(examples=
raw_examples, labels=raw_labels)

# evaluation with cosine similarity
itemtoitem = ItemToItem(ratings, movies, metric='cosine')
```

```
print("==================")
itemtoitem.evaluate(x_test, y_test)
```

Hasil yang diperoleh:

```
Normalize ratings ...
Create the similarity model ...
Compute nearest neighbors ...
Item to item recommendation model created with success .
==================
Evaluate the model on 10000 test data ...

MAE : 0.507794195659005
0.507794195659005
```

```
# evaluation with Euclidean distance

itemtoitem = ItemToItem(ratings, movies, metric='euclidean')
print("==================")
itemtoitem.evaluate(x_test, y_test)
```

Hasil yang diperoleh:

```
Normalize ratings ...
Create the similarity model ...
Compute nearest neighbors ...
Item to item recommendation model created with success ...
==================
Evaluate the model on 10000 test data ...

MAE : 0.8277111416143341
0.8277111416143341
```

c. Matrix Factorization

```
#Banyak iterasi
epochs = 10
```

```
# load the ml100k dataset
ratings, movies = ml100k.load()

ratings, uencoder, iencoder = ids_encoder(ratings)

m = ratings.userid.nunique()    # total number of users
n = ratings.itemid.nunique()    # total number of items

# get examples as tuples of userids and itemids and labels from
 normalize ratings
raw_examples, raw_labels = get_examples(ratings)

# train test split
```

```
(x_train, x_test), (y_train, y_test) = train_test_split(example
s=raw_examples, labels=raw_labels)

# create and train the model
mf = MF(m, n, k=10, alpha=0.01, lamb=1.5)

# fit the model on the training set
history = mf.fit(x_train, y_train, epochs=epochs, validation_da
ta=(x_test, y_test))
```

Hasil yang diperoleh:

```
Training Matrix Factorization Model ...
k=10     alpha=0.01      lambda=1.5
epoch 1/10 - loss : 2.734 - val_loss : 2.779
epoch 2/10 - loss : 1.764 - val_loss : 1.794
epoch 3/10 - loss : 1.592 - val_loss : 1.614
epoch 4/10 - loss : 1.538 - val_loss : 1.556
epoch 5/10 - loss : 1.515 - val_loss : 1.531
epoch 6/10 - loss : 1.503 - val_loss : 1.517
epoch 7/10 - loss : 1.496 - val_loss : 1.509
epoch 8/10 - loss : 1.491 - val_loss : 1.504
epoch 9/10 - loss : 1.488 - val_loss : 1.5
epoch 10/10 - loss : 1.486 - val_loss : 1.497
```

d. Non-Negative Matrix Factorization

```
from surprise import NMF
from surprise import Dataset
from surprise.model_selection import cross_validate

# Load the movielens-100k dataset (download it if needed).
data = Dataset.load_builtin('ml-100k')

# Use the NMF algorithm.
nmf = NMF(n_factors=10, n_epochs=10)

# Run 5-fold cross-validation and print results.
history = cross_validate(nmf, data, measures=['MAE'], cv=5, ver
bose=True)
```

Hasil yang diperoleh:

```
Evaluating MAE of algorithm NMF on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
MAE (testset)     0.9615  0.9501  0.9548  0.9582  0.9675  0.9584  0.0059
Fit time          0.67    0.72    0.79    0.72    0.72    0.72    0.04
Test time         0.15    0.09    0.16    0.10    0.14    0.13    0.03
```

e. Explainable Matrix Factorization

```
# load data
ratings, movies = ml100k.load()

# encode users and items ids
ratings, uencoder, iencoder = ids_encoder(ratings)

users = sorted(ratings.userid.unique())
items = sorted(ratings.itemid.unique())

m = len(users)
n = len(items)

# get examples as tuples of userids and itemids and labels from
 normalize ratings
raw_examples, raw_labels = get_examples(ratings)

# train test split
(x_train, x_test), (y_train, y_test) = train_test_split(example
s=raw_examples, labels=raw_labels)

# create the user to user model for similarity measure
usertouser = UserToUser(ratings, movies)

# compute explainable score
W = explainable_score(usertouser, users, items)

print("====================")
# initialize the model
emf = EMF(m, n, W, alpha=0.01, beta=0.4, lamb=0.01, k=10)

history = emf.fit(x_train, y_train, epochs=epochs, validation_d
ata=(x_test, y_test))

print("====================")
emf.evaluate(x_test, y_test)
```

Hasil yang diperoleh:

```
Normalize users ratings ...
Initialize the similarity model ...
Compute nearest neighbors ...
User to user recommendation model created with success ...
Compute explainable scores ...
====================
Training EMF
k=10     alpha=0.01      beta=0.4        lambda=0.01
epoch 1/10 - loss : 0.922 - val_loss : 1.036
epoch 2/10 - loss : 0.79 - val_loss : 0.873
epoch 3/10 - loss : 0.766 - val_loss : 0.837
epoch 4/10 - loss : 0.757 - val_loss : 0.822
epoch 5/10 - loss : 0.753 - val_loss : 0.814
epoch 6/10 - loss : 0.751 - val_loss : 0.808
epoch 7/10 - loss : 0.749 - val_loss : 0.805
epoch 8/10 - loss : 0.748 - val_loss : 0.802
epoch 9/10 - loss : 0.746 - val_loss : 0.799
epoch 10/10 - loss : 0.745 - val_loss : 0.797
====================
MAE : 0.797
0.7973478247232839
```

4. Result On Movielens 1M (ML-1M)
   a. Used-Based CF

```
# load ml100k ratings
ratings, movies = ml1m.load()

# prepare data
ratings, uencoder, iencoder = ids_encoder(ratings)

# get examples as tuples of userids and itemids and labels from
 normalize ratings
raw_examples, raw_labels = get_examples(ratings, labels_column=
'rating')

# train test split
(x_train, x_test), (y_train, y_test) = train_test_split(example
s=raw_examples, labels=raw_labels)

# metric : cosine

# create the user-based CF
usertouser = UserToUser(ratings, movies, k=20, metric='cosine')

# evaluate the user-based CF on the ml1m test data
print("==========================")
usertouser.evaluate(x_test, y_test)
```

Hasil yang diperoleh:

```
Normalize users ratings ...
Initialize the similarity model ...
Compute nearest neighbors ...
User to user recommendation model created with success ...
==========================
Evaluate the model on 100021 test data ...

MAE : 0.732267005840993
0.732267005840993
```

```
# metric : euclidean


# create the user-based CF
usertouser = UserToUser(ratings, movies, k=20, metric='euclide
   an')

# evaluate the user-based CF on the ml1m test data
print("==========================")
usertouser.evaluate(x_test, y_test)
```

Hasil yang diperoleh:

```
Normalize users ratings ...
Initialize the similarity model ...
Compute nearest neighbors ...
User to user recommendation model created with success ...
==========================
Evaluate the model on 100021 test data ...

MAE : 0.8069332535426615
0.8069332535426615
```

b. Item-Based CF

```
#cosine Similarity
itemtoitem = ItemToItem(ratings, movies, metric='cosine')
print("==========================")
itemtoitem.evaluate(x_test, y_test)
```

Hasil yang diperoleh:

```
Normalize ratings ...
Create the similarity model ...
Compute nearest neighbors ...
Item to item recommendation model created with success ...
==========================
Evaluate the model on 100021 test data ...

MAE : 0.42514728655396045
0.42514728655396045
```

```
#Euclidean Distance
itemtoitem = ItemToItem(ratings, movies, metric='euclidean')
print("==========================")
itemtoitem.evaluate(x_test, y_test)
```

Hasil yang diperoleh:

```
Normalize ratings ...
Create the similarity model ...
Compute nearest neighbors ...
Item to item recommendation model created with success ...
==========================
Evaluate the model on 100021 test data ...

MAE : 0.82502173206615
0.82502173206615
```

c. Matrix Factorization

```
# load the ml1m dataset
ratings, movies = ml1m.load()

ratings, uencoder, iencoder = ids_encoder(ratings)

m = ratings.userid.nunique()    # total number of users
```

```
n = ratings.itemid.nunique()    # total number of items

# get examples as tuples of userids and itemids and labels from
 normalize ratings
raw_examples, raw_labels = get_examples(ratings)

# train test split
(x_train, x_test), (y_train, y_test) = train_test_split(example
s=raw_examples, labels=raw_labels)

# create the model
model = MF(m, n, k=10, alpha=0.01, lamb=1.5)

# fit the model on the training set
history = model.fit(x_train, y_train, epochs=epochs, validation
_data=(x_test, y_test))

print("====================")
model.evaluate(x_test, y_test)
```

Hasil yang diperoleh:

```
Training Matrix Factorization Model ...
k=10     alpha=0.01      lambda=1.5
epoch 1/10 - loss : 1.713 - val_loss : 1.718
epoch 2/10 - loss : 1.523 - val_loss : 1.526
epoch 3/10 - loss : 1.496 - val_loss : 1.498
epoch 4/10 - loss : 1.489 - val_loss : 1.489
epoch 5/10 - loss : 1.485 - val_loss : 1.486
epoch 6/10 - loss : 1.484 - val_loss : 1.484
epoch 7/10 - loss : 1.483 - val_loss : 1.483
epoch 8/10 - loss : 1.483 - val_loss : 1.483
epoch 9/10 - loss : 1.482 - val_loss : 1.482
epoch 10/10 - loss : 1.482 - val_loss : 1.482
====================
validation error : 1.482
1.4820034560467208
```

d. Non-negative Matrix Factorization

```
from surprise import NMF
from surprise import Dataset
from surprise.model_selection import cross_validate

# Load the movielens-100k dataset (download it if needed).
data = Dataset.load_builtin('ml-1m')

# Use the NMF algorithm.
nmf = NMF(n_factors=10, n_epochs=10)

# Run 5-fold cross-validation and print results.
```

```
history = cross_validate(nmf, data, measures=['MAE'], cv=5, ver
bose=True)
```

Hasil yang diperoleh:

```
Evaluating MAE of algorithm NMF on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
MAE (testset)     0.9435  0.9456  0.9527  0.9546  0.9524  0.9498  0.0044
Fit time          6.35    6.51    6.52    6.52    6.55    6.49    0.07
Test time         1.20    1.47    1.46    1.47    1.47    1.42    0.11
```

e. Explainable Matrix Factorization

```
# load data
ratings, movies = ml1m.load()

# encode users and items ids
ratings, uencoder, iencoder = ids_encoder(ratings)

users = sorted(ratings.userid.unique())
items = sorted(ratings.itemid.unique())

m = len(users)
n = len(items)

# get examples as tuples of userids and itemids and labels from
 normalize ratings
raw_examples, raw_labels = get_examples(ratings)

# train test split
(x_train, x_test), (y_train, y_test) = train_test_split(example
s=raw_examples, labels=raw_labels)

# create the user to user model for similarity measure
usertouser = UserToUser(ratings, movies)

# compute explainable score
W = explainable_score(usertouser, users, items)

# initialize the model
emf = EMF(m, n, W, alpha=0.01, beta=0.4, lamb=0.01, k=10)

history = emf.fit(x_train, y_train, epochs=epochs, validation_d
ata=(x_test, y_test))
```

Hasil yang diperoleh:

```
Normalize users ratings ...
Initialize the similarity model ...
Compute nearest neighbors ...
User to user recommendation model created with success ...
Compute explainable scores ...
Training EMF
k=10     alpha=0.01      beta=0.4        lambda=0.01
epoch 1/10 - loss : 0.782 - val_loss : 0.807
epoch 2/10 - loss : 0.762 - val_loss : 0.781
epoch 3/10 - loss : 0.76 - val_loss : 0.775
epoch 4/10 - loss : 0.758 - val_loss : 0.771
epoch 5/10 - loss : 0.757 - val_loss : 0.769
epoch 6/10 - loss : 0.756 - val_loss : 0.767
epoch 7/10 - loss : 0.754 - val_loss : 0.764
epoch 8/10 - loss : 0.752 - val_loss : 0.762
epoch 9/10 - loss : 0.751 - val_loss : 0.761
epoch 10/10 - loss : 0.75 - val_loss : 0.76
```

Kesimpulan:

Summary

MAE comparison between User-based and Item-based CF

Metric	Dataset	User-based	Item-based
Euclidean	ML-100k	0.81	0.83
Euclidean	ML-1M	0.81	0.82
Cosine	ML-100k	0.75	0.51
Cosine	ML-1M	0.73	0.42

MAE comparison between MF, NMF and EMF

Preprocessing	Dataset	MF	NMF	EMF
Raw data	ML-100k	1.497	0.951	0.797
Raw data	ML-1M	1.482	0.9567	0.76
Normalized data	ML-100k	0.828	—	0.783
Normalized data	ML-1M	0.825	—	0.758

Secara umum, mean absolute error (MAE) yang lebih rendah lebih baik daripada MAE yang lebih tinggi, karena berarti prediksi model lebih mendekati nilai-nilai yang benar.

Daftar Pustaka

[1]. Shodik, N., Neneng, N., & Ahmad, I. (2018). Sistem Rekomendasi Pemilihan Smartphone Snapdragon 636 Menggunakan Metode Simple Multi Attribute Rating Technique (Smart). *Jurnal Nasional Pendidikan Teknik Informatika: JANAPATI*, *7*(3), 219-228.

[2]. Irfan, M., & Cahyani, A. D. (2014). Sistem Rekomendasi: Buku Online Dengan Metode Collaborative Filtering. *Jurnal Teknologi Technoscientia*, 076-84.

[3]. Jannach, D., Zanker, M., Fuchs, M., & Friedrich, G. (2017). Recommender systems: An introduction. Cambridge University Press.

[4] Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. Dalam Prosiding Konferensi Dunia Maya ke-10 (hlm. 285-295). ACM.

[5] Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). A collaborative filtering algorithm based on the concept of political democracy. Expert Systems with Applications, 40(10), 3491-3499.

[6] Amatriain, X., & Balcan, M. F. (2014). Matrix factorization techniques for recommender systems. In Recommender Systems Handbook (pp. 143-186). Springer, Boston, MA.

[7] Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. Data Mining, IEEE Transactions on, 8(1), 15-28.

[8] Somekh, O. (2015). Collaborative filtering: A brief history and current state of the art. In Recommender Systems Handbook (pp. 1-36). Springer, Boston, MA.

[9] Resnick, P., & Varian, H. (1997). Sistem rekomendasi. Communications of the ACM, 40(3), 56-58.

[10] Herlocker, J. L., Konstan, J. A., Borchers, A., & Riedl, J. (1999). Kerangka algoritma untuk melakukan collaborative filtering. Dalam Prosiding Konferensi SIGIR ke-22 tahunan internasional ACM tentang Penelitian dan pengembangan di retrieval informasi (hlm. 230-237). ACM.

[11] Michael D. Ekstrand, et al. (2011). Collaborative Filtering Recommender Systems

Code Reproduce: https://github.com/nzhinusoftcm/review-on-collaborative-filtering

Akun github: https://github.com/Agustinaginting/Final-Exam-Komputasi-Lanjut