LAPORAN

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

Dosen Pengampu:
Prof. Alhadi Bustamam, S.Si., M.Kom., Ph.D.
Risman Adnan, M.Si.

disusun oleh:
Cintya Kusuma Mahadhika 2206014725

Program Magister Matematika
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Indonesia
2023

PENDAHULUAN

Ketika internet terus berkembang, semakin sulit untuk menyaring semua informasi yang tersedia. Dengan banyaknya data dan pilihan yang dapat dibuat, orang membutuhkan filter untuk meningkatkan kegunaan internet. Sebuah teknik yang digunakan untuk mengatasi masalah ini adalah *collaborative filtering* (juga dikenal sebagai penyaringan sosial), yang mengurangi waktu yang dihabiskan untuk mencari dan meningkatkan akurasi pengambilan (Mason, n.d.). Secara umum, *collaborative filtering* adalah proses pemfilteran informasi atau pola menggunakan teknik yang melibatkan kolaborasi di antara banyak agen, sudut pandang, maupun sumber data. Aplikasi collaborative filtering biasanya melibatkan kumpulan data yang sangat besar. Metode penyaringan kolaboratif telah diterapkan pada berbagai jenis data termasuk: penginderaan dan pemantauan data, seperti dalam eksplorasi mineral, penginderaan lingkungan di area yang luas atau beberapa sensor; data keuangan, seperti lembaga jasa keuangan yang mengintegrasikan banyak sumber keuangan; atau dalam perdagangan elektronik dan aplikasi web yang berfokus pada data pengguna (Terveen et. al. 2001).

Beberapa tipe *collaborative filtering* adalah sebagai berikut.

1. *Memory-based*. Pendekatan *memory-based* menggunakan data peringkat pengguna untuk menghitung kesamaan antara pengguna atau item. (Charu, 2016). Contoh dari *memory-based collaborative filtering* adalah *user-based collaborative filtering* dan *item-based collaborative filtering*,
2. *Model-based*. Melalui pendekatan ini, metode reduksi dimensi sebagian besar digunakan sebagai teknik pelengkap untuk meningkatkan ketahanan dan akurasi pendekatan berbasis memori. Dalam pengertian ini, metode seperti dekomposisi nilai singular, analisis komponen utama, yang dikenal sebagai model faktor laten, memampatkan matriks item-pengguna menjadi representasi dimensi rendah dalam hal faktor laten. Salah satu keuntungan menggunakan pendekatan ini adalah bahwa alih-alih memiliki matriks berdimensi tinggi yang mengandung banyak nilai yang hilang, kita akan berhadapan dengan matriks yang jauh lebih kecil di ruang berdimensi lebih rendah. Presentasi yang dikurangi dapat digunakan baik untuk algoritma lingkungan berbasis pengguna atau berbasis item yang disajikan di bagian sebelumnya. Ada beberapa keuntungan dengan paradigma ini. Ini menangani sparsity dari matriks asli lebih baik daripada yang berbasis memori. Juga

membandingkan kesamaan pada matriks yang dihasilkan jauh lebih terukur terutama dalam menangani dataset jarang yang besar (Charu, 2016). Dalam model ini, rating dinyatakan dalam matriks (biasanya dinamakan R). Dalam prosesnya, ada kalanya matriks R perlu didekomposisi menjadi matriks yang lebih sederhana. Beberapa metode untuk mendekomposisi matriks R antara lain *singular value decomposition,* algoritma *matrix factorization*, *non negative matrix factorization*, dan *explainable matrix factorization*

Berdasarkan pemaparan di atas, dalam makalah ini akan dijabarkan model u*ser-based collaborative filtering*, *item-based collaborative filtering*, *singular value decomposition,* algoritma *matrix factorization*, *non negative matrix factorization*, dan *explainable matrix factorization*. Diberikan pula perbandingan performa masing-masing model dan kode menggunakan Google Colab.

PEMAPARAN MODEL

7.1. Download And Explore Movielen Latest Small

Sebelum membuat model rekomendasi, data perlu diunduh dahulu dan dieksplorasi. Salah satu cara yang dapat digunakan adalah mengurutkan data item serta mengelompokkan data dengan peringkat tertinggi maupun terendah. Contoh eksplorasi data akan dipaparkan selanjutnya.

7.2. User-based Collaborative Filtering

Dalam tahap ini, akan dibuat filter berdasarkan selera user dengan langkah-langkah berikut (misalkan u adalah user yang hendak dibuat daftar rekomendasinya).

1, Temukan pengguna lain yang perilaku pemberian peringkat sebelumnya mirip dengan u

2. Gunakan peringkat mereka pada item lain untuk memprediksi apa yang disukai pengguna saat ini.

Adapun algoritma user-to-user *collaborative filtering* adalah sebagai berikut (J. Bobadilla et al. 2013):

1. Pertama identifikasi himpunan Gu dari k pengguna yang paling mirip. Gu adalah pengguna grup yang mirip dengan pengguna aktif u. Kemiripan antara dua pengguna u dan v dapat diukur dengan ukuran kesamaan kosinus sebagai berikut :

$$w_{u,v} = \frac{\vec{r}_u \cdot \vec{r}_v}{\|\vec{r}_u\|_2 * \|\vec{r}_v\|_2} = \frac{\sum_{i \in I} r_{u,i} r_{v,i}}{\sqrt{\sum_{i \in I} (r_{u,i})^2} \sqrt{\sum_{i \in I} (r_{v,i})^2}}$$

$w_{u,v}$ adalah tingkat kesamaan antara pengguna u dan v . Istilah ini dihitung untuk semua $v \in U$, di mana U adalah himpunan semua pengguna. Masih ada pertanyaan tentang berapa banyak tetangga yang harus dipilih. Seperti yang diuji oleh Herlocker et al. (1999), $k \in [20,50]$ adalah titik awal yang mungkin di banyak domain.

2. Temukan set C item kandidat, dibeli oleh grup dan tidak dibeli oleh pengguna aktif u . Item kandidat harus menjadi item yang paling sering dibeli oleh grup.

3. Peringkat agregat pengguna di *Gu* untuk membuat prediksi bagi pengguna *u* pada item yang belum dia beli. Beberapa pendekatan agregasi sering digunakan seperti rata-rata, jumlah tertimbang, jumlah tertimbang yang disesuaikan. Dengan menggunakan weighted sum, prediksi rating pengguna *u* pada item *i* yang dilambangkan dengan $\hat{r}_{u,i}$ dihitung sebagai berikut :

$$\hat{r}_{u,i} = \bar{r}_u + \frac{\sum_{v \in G_u} (r_{v,i} - \bar{r}_v) \cdot w_{u,v}}{\sum_{v \in G_u} |w_{u,v}|}.$$

Peringkat pengguna serupa ditimbang berdasarkan kesamaan yang sesuai dengan pengguna aktif. Penjumlahan dibuat untuk semua pengguna yang telah menilai item *i* . Mengurangi peringkat rata-rata pengguna $\bar{r}_v$ mengkompensasi perbedaan dalam penggunaan skala peringkat oleh pengguna karena beberapa pengguna cenderung memberikan peringkat lebih tinggi daripada yang lain (Michael D. Ekstrand, et al. 2011). Prediksi ini dibuat untuk semua item $i \in C$ yang tidak dibeli oleh pengguna *u*.

4. Rekomendasi Top- *N* diperoleh dengan memilih *N* item yang paling memberikan kepuasan kepada user menurut prediksi.

Model ini memiliki beberapa keterbatasan, antara lain:

1. Ketersebaran. Secara umum, pengguna berinteraksi dengan kurang dari 20% item. Hal ini menyebabkan matriks peringkat menjadi sangat jarang. Misalnya, movielen-100k berisi peringkat 100k dari 943 pengguna pada 1682 item. Persentase dari sparsity dalam hal ini adalah sekitar 94% . Sistem pemberi rekomendasi berdasarkan algoritma tetangga terdekat mungkin tidak dapat membuat rekomendasi item apa pun untuk pengguna tertentu. Akibatnya, akurasi rekomendasi mungkin buruk (Sarwar et al. 2001).

2. Stabilitas peringkat pengguna. Saat pengguna memberi peringkat dan menilai ulang item, vektor peringkat mereka akan berubah seiring dengan

kemiripannya dengan pengguna lain. Lingkungan pengguna ditentukan tidak hanya oleh peringkat mereka tetapi juga oleh peringkat pengguna lain, sehingga lingkungan mereka dapat berubah sebagai akibat dari peringkat baru yang diberikan oleh setiap pengguna dalam sistem (Michael D. Ekstrand, et al. 2011).

3. Skalabilitas. Karena peringkat pengguna yang tidak stabil, sulit untuk menemukan pengguna serupa sebelumnya. Untuk alasan ini, sebagian besar sistem CF berbasis pengguna menemukan lingkungan setiap kali prediksi atau rekomendasi diperlukan. Namun, ini adalah perhitungan besar yang tumbuh dengan jumlah pengguna dan jumlah item. Dengan jutaan pengguna dan item, sistem pemberi rekomendasi berbasis web biasa yang menjalankan algoritme yang ada akan mengalami masalah skalabilitas yang serius (Sarwar et al. 2001), (Michael D. Ekstrand, et al. 2011).

7.3. Item-based Collaborative Filtering

Dalam tahap ini, akan dibuat filter berdasarkan selera user terhadap item yang mirip dengan item yang ingin diketahui. Langkah-langkah untuk *item based collaborative filtering* adalah sebagai berikut (misalkan u adalah user yang hendak dibuat daftar rekomendasinya dan i adalah barang yang akan diprediksi rate usernya).

1. Jika u menyukai barang yang mirip dengan i , dia mungkin akan menyukai barang i .
2. Jika dia membenci atau tidak menyukai barang yang mirip dengan i , dia juga akan membenci barang i .

Kelebihan dari item-based collaborative filtering adalah sebagai berikut.

1. Stabilitas. Peringkat item lebih stabil daripada peringkat pengguna. Peringkat baru pada item tidak mungkin mengubah kesamaan antara dua item secara signifikan, terutama ketika item memiliki banyak peringkat (Michael D. Ekstrand, et al. 2011).
2. Skalabilitas. Dengan peringkat item yang stabil, masuk akal untuk

melakukan pra-perhitungan kemiripan antar item dalam matriks kemiripan item-item (kesamaan antar item dapat dihitung secara offline). Ini akan mengurangi masalah skalabilitas dari algoritma. (Sarwar et al. 2001), (Michael D. Ekstrand, et al. 2011).

Algoritma dari item-based collaborative filtering adalah sebagai berikut (B. Sarwar et al. 2001) (George Karypis 2001).

1. Pertama-tama identifikasikan *k* item yang paling mirip untuk setiap item dalam katalog dan catat kesamaan yang sesuai. Untuk menghitung kesamaan antara dua item kita dapat menggunakan Adjusted Cosine Similarity yang telah terbukti lebih efisien daripada ukuran kesamaan Cosine dasar yang digunakan untuk kolaboratif berbasis pengguna seperti yang dijelaskan oleh B. Sarwar et al. *2001). Jarak *Adjusted cosine* antara dua item *i* dan *j* dihitung sebagai berikut.

$$w_{i,j} = \frac{\sum_{u \in U}(r_{u,i} - \bar{r}_u)(r_{u,j} - \bar{r}_u)}{\sqrt{\sum_{u \in U}(r_{u,i} - \bar{r}_u)^2}\sqrt{\sum_{u \in U}(r_{u,j} - \bar{r}_u)^2}}$$

$w_{i,j}$ adalah tingkat kesamaan antara item *i* dan *j* . Istilah ini dihitung untuk semua pengguna $u \in U$, di mana *U* adalah himpunan pengguna yang menilai kedua item *i* dan *j* . Himpunan dari *k* item yang paling mirip dengan item *i* akan dinyatakan dengan $S^{(i)}$.

2. Untuk menghasilkan rekomendasi top-N untuk pengguna tertentu *u* yang telah membeli satu set item I_u, dapat dilakukan hal berikut.
    a. Menemukan himpunan kandidat item C dengan mengambil semua gabungan $S^{(i)}$ untuk setiap $i \in I_u$ dan menghapus masing-masing item dari himpunan I_u

$$C = \bigcup_{i \in I_u} \{S^{(i)}\} \setminus I_u$$

b. Menemukan kemiripan antara c dan I_u, di mana c adalah anggota himpunan C.

$$w_{c,I_u} = \sum_{i \in I_u} w_{c,i}, \forall c \in C$$

c. Mengurutkan item C dalam urutan menurun dan mengembalikan N item pertama sebagai daftar rekomendasi Top-N.

Sebelum mengembalikan N item pertama sebagai daftar rekomendasi N teratas, dapat dibuat prediksi tentang apa yang akan diberikan pengguna u untuk setiap item dalam daftar rekomendasi N teratas, mengatur ulang daftar dalam urutan menurun dari peringkat yang diprediksi dan mengembalikan daftar yang disusun ulang sebagai daftar rekomendasi akhir. Prediksi rating untuk item-based CF diberikan dengan formula berikut (B. Sarwar et al. 2001):

$$\hat{r}_{u,i} = \frac{\sum_{i \in S^{(i)}} r_{u,j} \cdot w_{i,j}}{\sum_{j \in S^{(i)}} |w_{i,j}|}$$

7.4. Singular Value Decomposition

Karena tingkat ketersebaran yang tinggi dari matriks peringkat R , pemfilteran kolaboratif berbasis pengguna dan berbasis item mengalami ketersebaran dan skalabilitas data. Ini menyebabkan collaborative filtering berbasis pengguna dan item menjadi kurang efektif dan sangat memengaruhi kinerjanya.

Untuk mengatasi masalah sparsity tingkat tinggi, Sarwar et al. (2000) mengusulkan untuk mengurangi dimensi rating R menggunakan algoritma Singular Value Decomposition (SVD). *Singular value decomposition* (SVD) didefinisikan sebagai berikut.

Diberikan matriks A berukuran m x n. Matriks A dapat difaktorisasi menjadi matriks $U \sum V^T$, di mana matriks U adalah matriks ortogonal berukuran m x m, $\sum$ adalah matriks diagonal berukuran m x n, dan V adalah matriks ortogonal berukuran n x n.

Langkah-langkah SVD adalah sebagai berikut.

1. Menentukan nilai eigen dari matriks $A^T A$.
2. Menentukan V^T masing-masing kolomnya diperoleh dari vektor eigen yang bersesuaian dengan nilai eigen matriks $A^T A$.
3. Menentukan matriks diagonal $\sum$ dengan entri diagonalnya adalah nilai eigen real dari $A^T A$.
4. Menentukan matriks U yang masing-masing kolomnya diperoleh dari vektor eigen yang bersesuaian dengan nilai eigen matriks AA^T.

Sebelum menerapkan SVD, penting untuk mengisi nilai yang hilang dari matriks peringkat R . Sarwar dkk. (2000) menemukan peringkat rata-rata item menjadi nilai default yang berguna. Peringkat rata-rata pengguna juga dapat digunakan tetapi yang pertama menunjukkan kinerja yang lebih baik. Menambahkan normalisasi peringkat dengan mengurangi peringkat rata-rata pengguna atau prediktor garis dasar lainnya dapat meningkatkan akurasi.

7.5. Matrix Factorization

Salah satu cara untuk mengurangi dimensi matriks rating R adalah dengan *matrix factorization* (MF). Matrix Factorization (MF) memetakan pengguna dan item ke ruang faktor laten gabungan dari dimensi k sedemikian rupa sehingga interaksi pengguna-item dimodelkan sebagai produk dalam di ruang itu (Yehuda Koren et al., 2009). MF kemudian menguraikan R dalam dua matriks sebagai berikut :

$$R = Q \top P$$

Dimana $P \in \mathbb{R} m \times k$ merepresentasikan faktor laten pengguna dan $Q \in \mathbb{R} n \times k$ merepresentasikan faktor laten item. Setiap baris P, katakanlah $p_u \in \mathbb{R} k$ menunjukkan selera pengguna u dan setiap $q_i \in \mathbb{R} k$ fitur item i. Perkalian titik (*dot*

product) antara p_u dan q_i akan menjadi prediksi rating pengguna *u* pada item *i* :

$$\hat{r}_{u,i} = q_i^\top p_u$$

Selanjutnya, p_u dan q_i disebut dengan faktor laten.
Untuk mempelajari faktor laten, sistem meminimalkan kesalahan kuadrat yang diatur pada himpunan rating yang diketahui. Selanjutnya, fungsi biaya *J* didefinisikan sebagai berikut:

$$J = \frac{1}{2} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^\top p_u)^2 + \lambda(||p_u||^2 + ||q_i||^2)$$

Di mana κ adalah himpunan pasangan (u,i) di mana $r_{u,i}$ merupakan training set, dan λ adalah parameter yang teregularisasi. Fungsi J inilah yang nanti akan diminimumkan.

Kekurangan dari MF adalah nilai *P* dan *Q* dalam MF tidak dapat ditafsirkan karena komponennya dapat mengambil nilai sembarang (positif dan negatif). Kekurangan ini dapat diatasi dengan mendekomposisi matriks menjadi matriks non negatif yang dijabarkan dalam poin 7.6.

7.6. Non Negative Matrix Factorization

NMF (Lee dan Seung, 1999) memungkinkan rekonstruksi *P* dan *Q* sedemikian rupa sehingga $P,Q \geq 0$. Membatasi nilai *P* dan *Q* diambil dari [0,1] akan memungkinkan interpretasi probabilistik dengan parameter berikut.

1. Faktor laten mewakili kelompok pengguna yang memiliki selera yang sama,
2. Nilai *Pu,l* mewakili probabilitas bahwa pengguna *u* termasuk dalam grup *l* pengguna dan
3. Nilai *Ql,i* mewakili probabilitas bahwa pengguna dalam grup *l* menyukai item *i* .

Fungsi obyektif NMF didefinisikan sebagai berikut.

$$J = \frac{1}{2} \sum_{(u,i)\in\kappa} ||R_{u,i} - P_u Q_i^\top||^2 + \lambda_P ||P_u||^2 + \lambda_Q ||Q_i||^2$$

Selanjutnya, menurut (Lee dan Seung, 1999), aturan pembaharuan perkalian untuk P dan Q adalah sebagai berikut.

$$P \leftarrow P \cdot \frac{RQ}{PQ^\top Q}$$

$$Q \leftarrow Q \cdot \frac{R^\top P}{QP^\top P}$$

Namun, karena R adalah *sparse matrix*, setiap P_u perlu diperbarui sesuai dengan peringkat pengguna u yang ada. Demikian pula, Q_I perlu diperbarui sesuai dengan rating yang ada pada item i. Sehingga, elemen P dan Q diperbarui sebagai berikut.

$$P_{u,k} \leftarrow P_{u,k} \cdot \frac{\sum_{i\in I_u} Q_{i,k} \cdot r_{u,i}}{\sum_{i\in I_u} Q_{i,k} \cdot \hat{r}_{u,i} + \lambda_P |I_u| P_{u,k}}$$

$$Q_{i,k} \leftarrow Q_{i,k} \cdot \frac{\sum_{u\in U_i} P_{u,k} \cdot r_{u,i}}{\sum_{u\in U_i} P_{u,k} \cdot \hat{r}_{u,i} + \lambda_Q |U_i| Q_{i,k}}$$

7.7. Explainable Matrix Factorization

Dalam algoritma Explainable Matrix Factorization (EMF), ditambahkan informasi eksternal ke dalam model untuk memberikan penjelasan. Proses ini dapat menghitung skor yang dapat dijelaskan dari kesamaan pengguna atau item yang diambil dari CF berbasis pengguna atau berbasis item. Proses mengkuantifikasi penjelasan ini dijabarkan sebagai berikut.

1. Gunakan distribusi peringkat dalam lingkungan pengguna aktif.
2. Jika banyak *neighbor* telah menilai item yang direkomendasikan, maka ini dapat memberikan dasar untuk menjelaskan rekomendasi tersebut dengan menggunakan mekanisme penjelasan *environment*.

Menurut Abdollahi dan Nasraoui (2016), item i dianggap dapat dijelaskan oleh pengguna u jika sejumlah besar tetangganya menilai item i . Skor keterjelasan *Eui* adalah persentase tetangga pengguna u yang telah menilai item i .

$$E_{ui} = \frac{|N_k^{(i)}(u)|}{|N_k(u)|},$$

di mana $N_k(u)$ adalah himpunan k tetangga terdekat dari pengguna u dan $N_k^{(i)}(u)$ adalah himpunan tetangga pengguna u yang telah menilai item i. Namun, hanya skor yang dapat dijelaskan di atas ambang optimal θ yang diterima.

Dengan memasukkan *explainable weight* dalam algoritma pelatihan, fungsi tujuan baru, yang akan diminimalkan pada rangkaian peringkat yang diketahui, telah dirumuskan oleh Abdollahi dan Nasraoui (2016) sebagai:

$$J = \sum_{(u,i)\in\kappa} (R_{ui} - \hat{R}_{ui})^2 + \frac{\beta}{2}(||P_u||^2 + ||Q_i||^2) + \frac{\lambda}{2}(P_u - Q_i)^2 W_{ui},$$

Nilai J akan dioptimumkan dengan memperbarui P_u dan Q_i dengan *stochastic gradient descent* sebagai berikut.

$$P_u \leftarrow P_u + \alpha\left(2(R_{u,i} - P_u Q_i^\top)Q_i - \beta P_u - \lambda(P_u - Q_i)W_{ui}\right)$$
$$Q_i \leftarrow Q_i + \alpha\left(2(R_{u,i} - P_u Q_i^\top)P_u - \beta Q_i + \lambda(P_u - Q_i)W_{ui}\right)$$

7.8. Performance Measure

Performa dari model yang dipaparkan di atas dapat dibanding performanya dengan menggunakan *mean absolute error* (MAE). MAE adalah ukuran kesalahan antara pengamatan berpasangan yang menyatakan fenomena yang sama. Contoh Y versus X termasuk perbandingan prediksi versus waktu yang diamati, waktu berikutnya versus waktu awal, dan satu teknik pengukuran versus teknik pengukuran alternatif. MAE dihitung sebagai berikut (Willmot, 2005).

$$\text{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} = \frac{\sum_{i=1}^{n} |e_i|}{n}.$$

Di mana n adalah banyaknya data, y_i adalah nilai prediksi data ke-i dan x_i adalah nilai

sebenarnya dari data ke-i.

HASIL DAN KODE

7.1. Download And Explore Movielen Latest Small

Dari langkah pertama, akan dimuat data film yang pernah ditonton kemudian disusun daftar film dari rating tertinggi hingga terendah dengan langkah-langkah sebagai berikut.

1. Memuat dataset.

```
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
```

```
--2023-01-03 03:46:07--  https://github.com/nzhinusoftcm/review-on-collaborative-filtering/raw/master/recsys.zip
Resolving github.com (github.com)... 140.82.112.4
Connecting to github.com (github.com)|140.82.112.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/nzhinusoftcm/review-on-collaborative-filtering/master/recsys.zip [following]
--2023-01-03 03:46:07--  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 0.05s

2023-01-03 03:46:07 (317 MB/s) - 'recsys.zip' saved [15312323/15312323]
```

```
[2] from recsys.datasets import mlLatestSmall #ada typo di Lasted -> Latest

    import matplotlib.pyplot as plt
    import pandas as pd
    import zipfile
    import urllib.request
    import sys
    import os
```

```
[3] ratings, movies = mlLatestSmall.load() #load data
```

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

2. Memvisualisasikan data dalam tabel berdasarkan rating (tabel pertama) dan genre (tabel kedua).

```
[4] ratings.head()
```

	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] movies.head() #judul dan genre
```

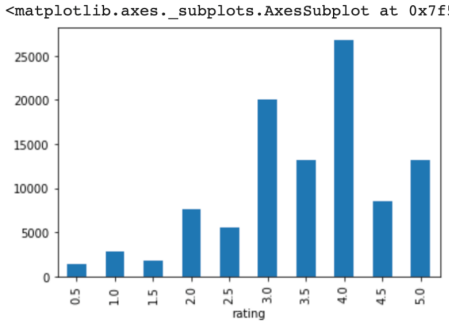
	itemid	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

3. Membuat histogram dari rating film.

Histogram of ratings

```
[6] ratings.groupby('rating').size().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5b6b93aa30>
```



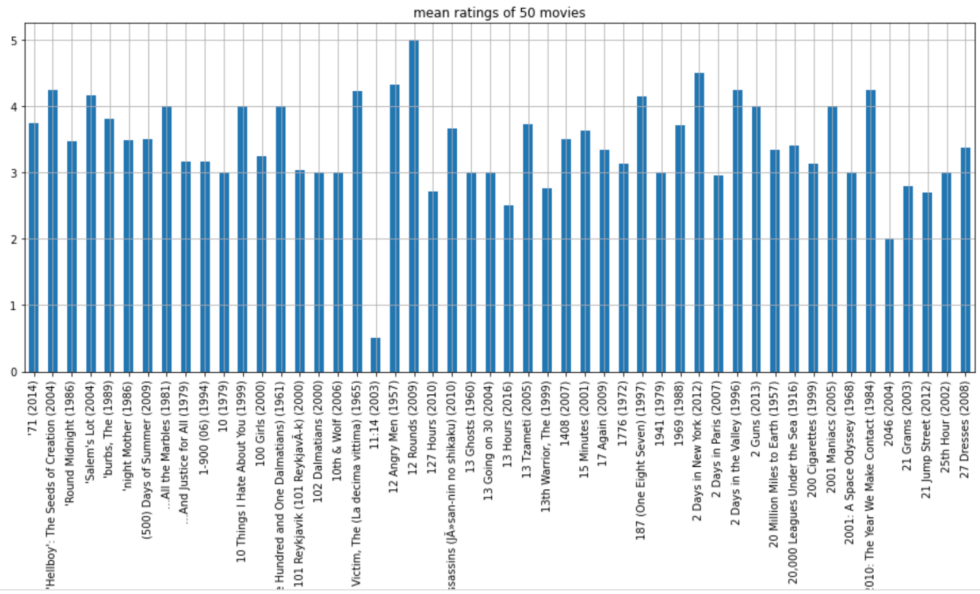
Peringkat berkisar dari $0,5$ hingga $5,0$, dengan langkah $0,5$. Histogram di atas menampilkan partisi ulang peringkat dalam kumpulan data. Dua peringkat yang paling banyak adalah 4.0 dan 3.0 dan peringkat yang paling sedikit adalah 0.5 dan 1.5

4. Menentukan rata-rata rating film.

Rata-Rata Rating Film

```
[7] movie_means = ratings.join(movies['title'], on='itemid').groupby('title').rating.mean()
    movie_means[:50].plot(kind='bar', grid=True, figsize=(16,6), title="mean ratings of 50 movies")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5b6b841ca0>
```

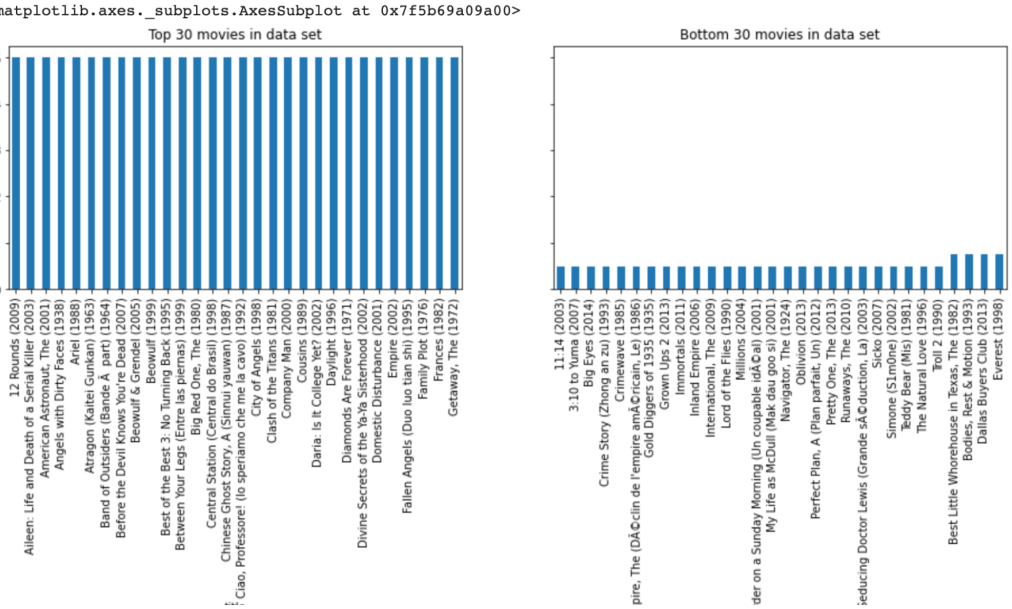


5. Membuat daftar 30 film dengan rating tertinggi dan 30 film dengan rating terendah.

30 Film dengan Rating Tertinggi vs. 30 Film dengan Rating Terendah

```
[8] fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(16,4), sharey=True)
    movie_means.nlargest(30).plot(kind='bar', ax=ax1, title="Top 30 movies in data set")
    movie_means.nsmallest(30).plot(kind='bar', ax=ax2, title="Bottom 30 movies in data set")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5b69a09a00>
```



7.2. User-based Collaborative Filtering

Dalam tahap ini, akan dibuat filter berdasarkan selera user dengan langkah-langkah berikut.

1. Memuat *tools* yang diperlukan.

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
```

```
Saving to: 'recsys.zip'

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

2023-01-03 04:05:37 (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
```

Import requirements

```
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
```

```
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
```

2. Memuat data

Load MovieLen ratings

```
[3] ratings, movies = ml100k.load()
```

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

userids and itemids encoding

```
# membuat encoder
ratings, uencoder, iencoder = ids_encoder(ratings)
```

3. Mengubah data ke dalam matriks

```
def ratings_matrix(ratings):
    return csr_matrix(pd.crosstab(ratings.userid, ratings.itemid, ratings.rating, aggfunc=sum).fillna(0).values)

R = ratings_matrix(ratings)
```

4. Membuat fungsi untuk menentukan jarak terdekat dari masing-masing user.

2. Function `nearest_neighbors()` returns the knn users for each user.

```
[9] def create_model(rating_matrix, metric): #membuar model nearest neighbors
        """
        - create the nearest neighbors model with the corresponding similarity metric
        - fit the model
        """
        model = NearestNeighbors(metric=metric, n_neighbors=21, algorithm='brute')
        model.fit(rating_matrix)
        return model
```

```
[10] 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 order of similarities
        """
        similarities, neighbors = model.kneighbors(rating_matrix)
        return similarities[:, 1:], neighbors[:, 1:]
```

Let's call functions `create_model()` and `nearest_neighbors()` to respectively create the k-NN model and compute the nearest neighbors for a given user

```
model = create_model(rating_matrix=R, metric='cosine') # membuat fungsi untuk menentukan jarak terdekat
similarities, neighbors = nearest_neighbors(R, model)
```

5. Membuat fungsi untuk menentukan *item* yang paling sering dibeli user atau kandidat.

```python
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)]

    # mengurutkan item dari frekuensi tertinggi
    frequency = activities.groupby('itemid')['rating'].count().reset_index(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=True)[:30]

    return candidates
```

6. Menentukan rata-rata rating untuk masing-masing user dan dinormalisasikan.

```python
[14] # rerata rating untuk masing-masing user
     mean = ratings.groupby(by='userid', as_index=False)['rating'].mean()
     mean_ratings = pd.merge(ratings, mean, suffixes=('','_mean'), on='userid')

     # normalized ratings untuk masing-masing item
     mean_ratings['norm_rating'] = mean_ratings['rating'] - mean_ratings['rating_mean']

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

```python
np_ratings = mean_ratings.to_numpy()
```

7. Membuat fungsi untuk memprediksi rating suatu item dari user.

```python
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]
    # mendapatkan rerata rating dari user
    user_mean = mean[userid]

    # menemukan user yang merating item 'itemid'
    iratings = np_ratings[np_ratings[:, 1].astype('int') == itemid]

    # menemukan user yang mirip dengan'userid' yang merating'itemid'
    suri = iratings[np.isin(iratings[:, 0], user_neighbors)]

    # user yang punya kemiripan yang merating 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.abs(sims))

    return r_hat
```

```python
def user2userPredictions(userid, pred_path):
    """
    Make rating prediction for the active user on each candidate item and save in file prediction.csv

    :param
        - userid : id of the active user
        - pred_path : where to save predictions
    """
    # menemukan kandidat item untuk user aktif
    candidates = find_candidate_items(userid)

    # membuat prediksi
    for itemid in candidates:

        # prediksi untuk userid pada itemid
        r_hat = predict(userid, itemid)

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

```python
[19] 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)
```

```python
user2userCF()
```

```
Rating predictions. Progress status : 99.9%
```

8. Membuat fungsi untuk membaca prediksi untuk pengguna tertentu dan menampilkan daftar item dalam urutan menurun dari peringkat yang

Step 4. Top-N recommendation

Function `user2userRecommendation()` reads predictions for a given user and return the list of items in decreasing order of predicted rating.

```
def user2userRecommendation(userid):
    """
    """
    # encode the userid
    uid = uencoder.transform([userid])[0]
    saved_predictions = 'predictions.csv'

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

    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
```

```
user2userRecommendation(212)
```

	userid	itemid	predicted_rating	title
0	212	88	5.108297	Sleepless in Seattle (1993)
1	212	167	5.011518	Private Benjamin (1980)
2	212	21	5.011458	Muppet Treasure Island (1996)
3	212	190	4.942885	Henry V (1989)
4	212	81	4.933844	Hudsucker Proxy, The (1994)

9. Melatih data dan mengecek model dengan *mean absolute error*.

```
[25] from recsys.preprocessing import train_test_split, get_examples

     # dapat contoh sebagai tupel userid dan itemid dan label dari normalisasi peringkat
     raw_examples, raw_labels = get_examples(ratings, labels_column='rating')

     # melatih data
     (x_train, x_test), (y_train, y_test) = train_test_split(examples=raw_examples, 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
```

```
evaluate(x_test, y_test)
```

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

MAE : 0.7505910931068639
0.7505910931068639
```

Dari program yang dijalankan, diperoleh MEA sekitar 75%.

7.3. Item-based Collaborative Filtering

Dalam tahap ini, akan diprediksi apakah user menyukai suatu item berdasarkan kesukaan (atau ketidaksukaan) user terhadap item yang mirip dengan item yang hendak diprediksi. Langkah-langkah yang dilakukan adalah sebagai berikut.

1. Mengimport *tools* yang diperlukan.

```
Run cell (⌘/Ctrl+Enter)
cell executed since last change

executed by Cintya Mahadhika
1:58 PM (0 minutes ago)
executed in 0.023s

exists("recsys.zip") or os.path.exists("recsys")):
//github.com/nzhinusoftcm/review-on-collaborative-filtering/raw/master/recsys.zip
s.zip
```

```
--2023-01-03 06:58:52--  https://github.com/nzhinusoftcm/review-on-collaborative-filtering/raw/master/recsys.zip
Resolving github.com (github.com)... 140.82.113.3
Connecting to github.com (github.com)|140.82.113.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/nzhinusoftcm/review-on-collaborative-filtering/master/recsys.zip [following]
--2023-01-03 06:58:52--  https://raw.githubusercontent.com/nzhinusoftcm/review-on-collaborative-filtering/master/recsys.zip
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133, 185.199.111.133, 185.199.109.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.133|: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 0.08s

2023-01-03 06:58:53 (184 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
```

```
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
```

2. Memuat data dan membuat *encoder*.

```
[3] ratings, movies = ml100k.load()
```

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

userids and itemids encoding

```
# membuat encoder
ratings, uencoder, iencoder = ids_encoder(ratings)
```

3. Menentukan kemiripan masing-masing item. Untuk langkah awal, didefinisikan fungsi untuk menormalisasikan mean rating terhadap masing-masing user.

```
[7] def normalize():
        # menghitung mean rating untuk masing-masing user
        mean = ratings.groupby(by='userid', as_index=False)['rating'].mean()
        norm_ratings = pd.merge(ratings, mean, suffixes=('','_mean'), on='userid')

        # menormalisasikan masing-masing rating dengan mengurangi mean rating terhadap masing-masing user
        norm_ratings['norm_rating'] = norm_ratings['rating'] - norm_ratings['rating_mean']
        return mean.to_numpy()[:, 1], norm_ratings
```

```
mean, norm_ratings = normalize()
np_ratings = norm_ratings.to_numpy()
norm_ratings.head()
```

	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

Selanjutnya, setiap item dinyatakan sebagai vektor hasil normalisasi.

Karena setiap rating sudah dinormalisasi, maka masing-masing item bisa direpresentasikan sebagai vektor hasil normalisasinya.

```
[9] def item_representation(ratings):
        return csr_matrix(
            pd.crosstab(ratings.itemid, ratings.userid, ratings.norm_rating, aggfunc=sum).fillna(0).values
        )
```

```
R = item_representation(norm_ratings)
```

Kemudian, dibuat model k-NN dengan sklearn.

Membuat k-NN model dengan 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
```

Langkah selanjutnya adalah menghitung kemiripan dengan jarak *cosine* antara dua vektor.

```
[12] 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:]
```

```
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)

# dapatkan neighbout dalam urutan kesamaan yang menurun
neighbors = np.flip(np.argsort(similarities), axis=1)

# mengurutkan kesamaan dalam urutan menurun
similarities = np.flip(np.sort(similarities), axis=1)

# menyimpan kemiripan dengan disk
save_similarities(similarities, neighbors, dataset_name=dataset_name)

return similarities, neighbors
```

Selanjutnya dapat dipilih filtering yang diinginkan.

```
[15] # metric : pilih [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)
```

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

5. Setelah fungsi untuk memfilter sudah didefinisikan, dibuat fungsi untuk memberi rekomendasi user berdasarkan perilaku user pada item yang mirip.

```
[17] def candidate_items(userid):
         """
         :param userid : user id for which we wish to find candidate items
         :return : I_u, candidates
         """

         # 1. Menemukan set I_u dari item yang sudah diberi peringkat oleh user id
         I_u = np_ratings[np_ratings[:, 0] == userid]
         I_u = I_u[:, 1].astype('int')

         # 2. Mengambil penyatuan item serupa untuk semua item di I_u untuk membentuk himpunan item kandidat
         c = set()

         for iid in I_u:
             # tambahkan tetangga item iid di set kandidat item
             c.update(neighbors[iid])

         c = list(c)
         # 3. kecualikan dari set C semua item di I_u.
         candidates = np.setdiff1d(c, I_u, assume_unique=True)

         return I_u, candidates
```

```
[18] 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))
```

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

```python
def similarity_with_Iu(c, I_u):
    """
    compute similarity between an item c and a set of items I_u. For each 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 nearest neighbors of itemid
        if c in neighbors[iid] :
            w = w + similarities[iid, neighbors[iid] == c][0]
    return w
```

2.c- Rank candidate items according to their similarities to I_u

```python
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 similarities 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], reverse=True)
    return ranked_candidates
```

6. Selanjutnya, akan disusun rekomendasi untuk user berdasarkan kesukaan user terhadap item yang serupa.

```python
[24] 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 recommendation list
        topn = pd.DataFrame(ranked_candidates[:N], columns=['itemid','similarity_with_Iu'])
        topn = pd.merge(topn, movies, on='itemid', how='inner')
        return topn
```

```python
topn_recommendation(test_user)
```

	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)

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...
20	1506	22.325504	Nelly & Monsieur Arnaud (1995)
21	1537	22.061914	Cosi (1996)
22	1474	21.877034	Nina Takes a Lover (1994)
23	1467	21.861203	Saint of Fort Washington, The (1993)
24	1255	21.750924	Broken English (1996)
25	1499	21.529748	Grosse Fatigue (1994)
26	1466	21.063269	Margaret's Museum (1995)
27	1448	20.846909	My Favorite Season (1993)
28	927	20.730153	Flower of My Secret, The (Flor de mi secreto, ...
29	1375	20.627152	Cement Garden, The (1993)

7. Sebelum memberikan rekomendasi seperti poin 6, rating user dapat diprediksi dengan fungsi berikut.

```
[1] 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 similarities
        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
```

```
def topn_prediction(userid):
    """
    :param userid : id of the active user
    :return topn : initial topN recommendations returned by the function 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','prediction'])

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

    return topn, topn_predict
```

8. Setelah itu, dapat dibuat rekomendasi untuk user sebagai berikut.

```
[24] topn, topn_predict = topn_prediction(userid=test_user)
```

```
topn_predict
```

	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
15	1485	24.345312	Colonel Chabert, Le (1994)	3.610294
11	1664	25.327445	8 Heads in a Duffel Bag (1997)	3.610294
9	691	26.461802	Dark City (1998)	3.610294

9. Selanjutnya, hasil prediksi akan dievaluasi dengan *mean absolute error* dan diperoleh hasil sekitar 67%.

```
[26] from recsys.preprocessing import train_test_split, get_examples

     # 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(examples=raw_examples, 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
```

```
evaluate(x_test, y_test)
```

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

MAE : 0.672389703640273
0.672389703640273
```

10. Langkah lain untuk mendapatkan user-based filtering adalah dengan menggunakan ItemToItem sebagai berikut.

```
[28] from recsys.memories.ItemToItem import ItemToItem
     from recsys.preprocessing import ids_encoder, train_test_split, get_examples
     from recsys.datasets import ml100k

     # 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 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(examples=raw_examples, labels=raw_labels)
```

Diperoleh *mean absolute error* 50%.

```
[29] # membuat Item-based CF
     item2item = ItemToItem(ratings, movies, k=20, metric='cosine', dataset_name='ml100k')

     Normalize ratings ...
     Create the similarity model ...
     Compute nearest neighbors ...
     Item to item recommendation model created with success ...

     # evaluasi algoritma
     item2item.evaluate(x_test, y_test)

     Evaluate the model on 10000 test data ...

     MAE : 0.507794195659005
     0.507794195659005
```

11. Adapun user-based filtering pada ML-1M dataset dapat dilakukan dengan kode berikut.

7.4. Singular Value Decomposition

Karena tingkat ketersebaran yang tinggi dari matriks prating R , collaborative filtering berbasis pengguna dan berbasis item mengalami ketersebaran dan skalabilitas data. Ini menyebabkan collaborative filtering berbasis pengguna dan item menjadi kurang efektif dan sangat memengaruhi kinerjanya.

Untuk mengatasi masalah sparsity tingkat tinggi, Sarwar et al. (2000) mengusulkan untuk mengurangi dimensi rating R menggunakan algoritma Singular Value Decomposition (SVD). Langkah-langkah yang diperoleh untuk membuat rekomendasi pengguna dengan matriks rating yang ditransformasi dengan SVD adalah sebagai berikut.

1. Memuat *tools* yang diperlukan.

```
[1] 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
```

```
Saving to: 'recsys.zip'

recsys.zip          100%[===================>]  14.60M  --.-KB/s    in 0.08s

2023-01-04 00:40:16 (176 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/
```

```
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
```

2. Memuat data rating film.

```
[4] ratings, movies = mlLatestSmall.load() #memuat data rating
```

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

Let's see how our rating matrix looks like

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

itemid	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573	193579	193581	193583	193585	193587	193609
userid																					
1	4.0	NaN	4.0	NaN	NaN	4.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	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	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	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	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
606	2.5	NaN	NaN	NaN	NaN	NaN	2.5	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
607	4.0	NaN	NaN	NaN	NaN	NaN	NaN	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	NaN	NaN	NaN	NaN	NaN	NaN
609	3.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
610	5.0	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

610 rows × 9724 columns

3. Mencari rerata rating user dan mengonversi data rating ke array.

```
[7] # mencari rerata rating user
    umean = ratings.groupby(by='userid')['rating'].mean()
```

```
[8] 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
        """

        # mengisi missing value dengan rerata
        df = pd.crosstab(ratings.userid, ratings.itemid, ratings.rating, aggfunc=sum)
        df = df.fillna(df.mean(axis=0))

        # mengurangi peringkat rata-rata pengguna untuk normalisasi data
        df = df.subtract(umean, axis=0)

        # konversi dataframe ke numpy array
        R = df.to_numpy()

        return R, df

    # menghasilkan matriks peringkat dengan memanggil fungsi rating_matrix
    R, df = rating_matrix(ratings)
```

```
df
```

itemid	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573	193579	19358
userid																	
1	-0.366379	-0.934561	-0.366379	-2.009236	-1.294951	-0.366379	-1.181194	-1.491379	-1.241379	-0.870167	...	-0.866379	-1.366379	-0.366379	-0.366379	-0.866379	-0.36637
2	-0.027346	-0.516458	-0.688660	-1.591133	-0.876847	-0.002197	-0.763091	-1.073276	-0.823276	-0.452064	...	-0.448276	-0.948276	0.051724	0.051724	-0.448276	0.05172
3	1.485033	0.995921	0.823718	-0.078755	0.635531	1.510181	0.749288	0.439103	0.689103	1.060315	...	1.064103	0.564103	1.564103	1.564103	1.064103	1.5641
4	0.365375	-0.123737	-0.295940	-1.198413	-0.484127	0.390523	-0.370370	-0.680556	-0.430556	-0.059343	...	-0.055556	-0.555556	0.444444	0.444444	-0.055556	0.44444
5	0.363636	-0.204545	-0.376748	-1.279221	-0.564935	0.309715	-0.451178	-0.761364	-0.511364	-0.140152	...	-0.136364	-0.636364	0.363636	0.363636	-0.136364	0.36363
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
606	-1.157399	-0.225581	-0.397784	-1.300256	-0.585971	0.288679	-1.157399	-0.782399	-0.532399	-0.161187	...	-0.157399	-0.657399	0.342601	0.342601	-0.157399	0.34260
607	0.213904	-0.354278	-0.526481	-1.428953	-0.714668	0.159982	-0.600911	-0.911096	-0.661096	-0.289884	...	-0.286096	-0.786096	0.213904	0.213904	-0.286096	0.2139
608	-0.634176	-1.134176	-1.134176	-0.777033	-0.062747	0.811903	0.051009	-0.259176	-0.009176	0.865824	...	0.365824	-0.134176	0.865824	0.865824	0.365824	0.8658
609	-0.270270	0.161548	-0.010655	-0.913127	-0.198842	0.675808	-0.085085	-0.395270	-0.145270	0.729730	...	0.229730	-0.270270	0.729730	0.729730	0.229730	0.7297
610	1.311444	-0.256738	-0.428941	-1.331413	-0.617127	1.311444	-0.503371	-0.813556	-0.563556	-0.192344	...	-0.188556	-0.688556	0.311444	0.311444	-0.188556	0.3114

610 rows × 9724 columns

4. Membuat item id sehingga diperoleh nilai 0-909 (untuk user) dan 0-9723 (untuk

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

# membuat id encodercreate our id encoders
uencoder = LabelEncoder()
iencoder = LabelEncoder()

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

```
LabelEncoder()
```

5. Membuat fungsi untuk transformasi SVD dan mentransformasi matriks rating.

```
[13] class SVD:

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

             # matriks resultan svd
             self.P = np.array([])
             self.S = np.array([])
             self.Qh = np.array([])

             # faktor laten user dan item
             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

             # faktor laten user (u_factors) dan 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 id user dan item
    u = uencoder.transform([userid])[0]
    i = iencoder.transform([itemid])[0]

    # prediksi rating adalah dot product antara barus ke uth
    # dari u_factors dan ke ith column dari i_factors
    r_hat = np.dot(self.u_factors[u,:], self.i_factors[:,i])

    # tambahkan peringkat rata-rata pengguna u ke nilai prediksi
    r_hat += self.umean[u]

    return r_hat
```

```python
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_factors returns
    # the predicted value for user u on all items
    predictions = np.dot(self.u_factors[u,:], self.i_factors) + self.umean[u]

    # mengurutkan id item dalam urutan penurunan prediksi
    top_idx = np.flip(np.argsort(predictions))

    # decode indices untuk memperoleh itemid
    top_items = iencoder.inverse_transform(top_idx)

    # sorted predictions
    preds = predictions[top_idx]

    return top_items, preds
```

```python
# membuat svd model
svd = SVD(umean)

# mencocokkab model dengan rating yang dinornalisasi
svd.fit(R)
```

6. Membuat prediksi dari matriks rating yang telah ditransformasikan.

```python
[17] ratings.head(10)
```

	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

```python
# user yang diprediksi
userid = 1

# daftar item untuk user 1
items = [1,3,6,47,50,70,101,110,151,157]

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

```
prediction for userid=1 and itemid=1 : 3.9999999999999996
prediction for userid=1 and itemid=3 : 4.0000000000000036
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 : 4.9999999999999964
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 : 5.0000000000000115
prediction for userid=1 and itemid=157 : 5.000000000000028
```

Our prediction error is less than 0.00001

Diperoleh kesalahan prediksi kurang dari 0.00001

7. Membuat fungsi yang dapat memberikan rekomendasi untuk user dan memberi rekomendasi untuk user terkait.

```python
userid = 1

# item diurutkan dalam urutan penurunan prediksi untuk pengguna 1
sorted_items, preds = svd.recommend(userid=userid)

##
# mengecualikan dari item daftar terurut yang sudah dibeli oleh pengguna
##

# daftar item yang dinilai oleh pengguna
uitems = ratings.loc[ratings.userid == userid].itemid.to_list()

# hapus dari item yang disortir item yang sudah ada di item dan pilih 30 item teratas
# sebagai daftar rekomendasi
top30 = np.setdiff1d(sorted_items, uitems, assume_unique=True)[:30]

# dapatkan prediksi yang sesuai dari 30 item teratas
top30_idx = list(np.where(sorted_items == idx)[0][0] for idx in top30)
top30_predictions = preds[top30_idx]

# temukan judul film yang sesuai
zipped_top30 = list(zip(top30,top30_predictions))
top30 = pd.DataFrame(zipped_top30, columns=['itemid','predictions'])
List = pd.merge(top30, movies, on='itemid', how='inner')

# menunjukkan daftar
List
```

	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
20	59814	5.0	Ex Drummer (2007)	Comedy|Crime|Drama|Horror
21	5745	5.0	Four Seasons, The (1981)	Comedy|Drama
22	118894	5.0	Scooby-Doo! Abracadabra-Doo (2010)	Animation|Children|Mystery
23	5746	5.0	Galaxy of Terror (Quest) (1981)	Action|Horror|Mystery|Sci-Fi
24	118834	5.0	National Lampoon's Bag Boy (2007)	Comedy
25	3940	5.0	Slumber Party Massacre III (1990)	Horror
26	95311	5.0	Presto (2008)	Animation|Children|Comedy|Fantasy
27	3496	5.0	Madame Sousatzka (1988)	Drama
28	156025	5.0	Ice Age: The Great Egg-Scapade (2016)	Adventure|Animation|Children|Comedy
29	2196	5.0	Knock Off (1998)	Action

7.5. Matrix Factorization

Algoritma Matrix Factorization adalah varian dari SVD. Juga dikenal sebagai Regularized SVD, di mana digunakan pengoptimal Gradient Descent untuk mengoptimalkan fungsi *cost* saat melatih model.

Langkah-langkah yang dilakukan adalah sebagai berikut.

1. Memuat *tools* yang diperlukan.

```
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
```

```
--2023-01-04 01:01:24--  https://github.com/nzhinusoftcm/review-on-collaborative-filtering/raw/master/recsys.zip
Resolving github.com (github.com)... 140.82.114.3
Connecting to github.com (github.com)|140.82.114.3|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/nzhinusoftcm/review-on-collaborative-filtering/master/recsys.zip [following]
--2023-01-04 01:01:24--  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

2023-01-04 01:01:25 (107 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
```

```
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
```

2. Mendefinisikan model.

```python
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)

        # inisialisasi matriks faktor laten P dan Q (masing-masing bentuk (m,k) dan (n,k)) yang akan dipelajari
        self.k = k
        self.P = np.random.normal(size=(m, k))
        self.Q = np.random.normal(size=(n, k))

        # inisialisasi hiperparameter
        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}')
```

```python
    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
        """
        # contoh banyaknya training
        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))
```

```python
    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 known
            - 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()

        # validasi data
        x_test, y_test = validation_data

        # loop banyaknya epoch epochs
        for epoch in range(1, epochs+1):

            # untuk setiap pasangan (u,i) dan peringkat r yang sesuai
            for pair, r in zip(x_train, y_train):

                # dapatkan nilai userid dan itemid yang disandikan dari pasangan
                u,i = pair

                # hitung prediksi rating r_hat
                r_hat = np.dot(self.P[u], self.Q[i])

                # menghitung kesalahan prediksi
                e = abs(r - r_hat)

                # perbarui
                self.update_rule(u, i, e)
```

```python
            # training dan validasi setelah 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)

            # cetak kemajuan pelatihan setelah setiap periode langkah
            self.print_training_progress(epoch, epochs, error, val_error, steps=1)

            # leaning rate scheduler : ulangi kecepatan pembelajaran saat kita masuk lebih dalam ke jumlah epoch
            # 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
```

```python
    def predict(self, userid, itemid):
        """
        Make rating prediction for a user on an item
        :param userid
        :param itemid
        :return r : predicted rating
        """
        # menyandikan id pengguna dan item agar dapat mengakses faktor latennya
         # matriks P dan Q
        u = uencoder.transform([userid])[0]
        i = iencoder.transform([itemid])[0]

        #prediksi peringkat menggunakan id yang disandikan (encoded). Dot product antara P_u dan 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
        with their corresponding predictions
        """
        # encode userid
        u = uencoder.transform([userid])[0]

        # prediksi untuk pengguna userid pada semua produk
        predictions = np.dot(self.P[u], self.Q.T)

        # dapatkan indeks prediksi N teratas
        top_idx = np.flip(np.argsort(predictions))[:N]

        # decode indeks untuk mendapatkan itemid yang sesuai
        top_items = iencoder.inverse_transform(top_idx)
```

```python
# ambil prediksi yang sesuai untuk indeks N teratas
preds = predictions[top_idx]

return top_items, preds
```

```python
epochs = 10
```

3. Mengevaluasi rating yang belum dinormalisasi untuk Movielens 100k.

```
[5] # memuat ml100k dataset
    ratings, movies = ml100k.load()

    ratings, uencoder, iencoder = ids_encoder(ratings)

    m = ratings.userid.nunique()   # banyaknya users
    n = ratings.itemid.nunique()   # banyaknya items

    # dapatkan contoh sebagai tupel userid dan itemid dan label dari normalisasi peringkat
    raw_examples, raw_labels = get_examples(ratings)

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

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

```
[6] # membuat 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_test, y_test))

    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)

validation error : 1.497
1.4973507972141993
```

Diperoleh *validation error* 1.497

4. Mengevaluasi rating yang dinormalisasikan untuk Movielens 100k.

```
[8] # memuat 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

    # menormalkan peringkat dengan cara mengurangi rerata
    normalized_column_name = "norm_rating"
    ratings = normalized_ratings(ratings, norm_column=normalized_column_name)

    # dapatkan contoh sebagai tupel userid dan itemid dan label dari normalisasi peringkat
    raw_examples, raw_labels = get_examples(ratings, labels_column=normalized_column_name)

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

```
[9] # membuat 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_test, y_test))
```

```
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
```

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

```
validation error : 0.828
0.8276982643684648
```

Diperoleh *validation error* 0.828.

5. Mengevaluasi data yang belum dinormalisasi untuk Movielens 1M.

```
[11] # 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(examples=raw_examples, labels=raw_labels)
```

```
Download data 100.1%
Successfully downloaded ml-1m.zip 5917549 bytes.
Unzipping the ml-1m.zip zip file ...
```

```
[12] # membuat 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_test, y_test))
```

```
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
```

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

```
validation error : 1.482
1.4820034560467208
```

Diperoleh *validation error* 1.482

6. Mengevaluasi data yang sudah dinormalisasi untuk Movielens 1M.

```
[14] # memuat data
     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

     # menormalisasi ratings dengan mengurangi means
     normalized_column_name = "norm_rating"
     ratings = normalized_ratings(ratings, norm_column=normalized_column_name)

     #dapatkan contoh sebagai tupel userid dan itemid dan label dari normalisasi peringkat
     raw_examples, raw_labels = get_examples(ratings, labels_column=normalized_column_name)

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

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

# sesuaikan model pada set pelatihan
history = MF.fit(x_train, y_train, epochs=epochs, validation_data=(x_test, y_test))
```

```
Training Matrix Factorization Model ...
k=10     alpha=0.01      lambda=1.5
epoch 1/10 - loss : 0.826 - val_loss : 0.827
epoch 2/10 - loss : 0.824 - val_loss : 0.825
epoch 3/10 - loss : 0.823 - val_loss : 0.825
epoch 4/10 - loss : 0.823 - val_loss : 0.825
epoch 5/10 - loss : 0.823 - val_loss : 0.825
epoch 6/10 - loss : 0.823 - val_loss : 0.825
epoch 7/10 - loss : 0.823 - val_loss : 0.825
epoch 8/10 - loss : 0.823 - val_loss : 0.825
epoch 9/10 - loss : 0.823 - val_loss : 0.825
epoch 10/10 - loss : 0.823 - val_loss : 0.825
```

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

```
validation error : 0.825
0.8250208634455388
```

Diperoleh *validation error* 0.825.

7. Membuat prediksi

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

	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 have used the normalised ratings for training
```

```
4.188679163563357
```

7.6. Non Negative Matrix Factorization

Algoritma Non Negative Matrix Factorization mirip dengan Matrix Factorization, namun dalam algoritma ini matriks yang dihasilkan tidak memiliki elemen negatif.

Langkah-langkah yang dilakukan adalah sebagai berikut.

1. Memuat *tools* yang diperlukan.

```
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
```

```
Saving to: 'recsys.zip'

recsys.zip          100%[===================>]  14.60M  --.-KB/s    in 0.08s

2023-01-04 01:34:28 (175 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
```

```
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
```

2. Memuat rating

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

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

# konversi rating dari dataframe ke numpy array
np_ratings = ratings.to_numpy()

# dapatkan contoh sebagai tupel userid dan itemid dan label dari normalisasi peringkat
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_examples, labels=raw_labels)
```

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

3. Membuat model *non negative matrix factorization*

```python
class NMF:

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

        np.random.seed(32)

        # inisialisasi matriks faktor laten P dan Q (masing-masing bentuk (m,k) dan (n,k)) yang akan dipelajari
        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

        # menginisialisasi pembuat enkode
        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}')
```

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

```python
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)
        # kesalahan pelatihan dan validasi setelah zaman ini
        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_error, 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
```

```python
def recommend(self, userid, N=30):
    # menyandikan userid
    u = self.uencoder.transform([userid])[0]

    # prediksi untuk pengguna userid pada semua produk
    predictions = np.dot(self.P[u], self.Q.T)

    #dapatkan indeks prediksi N teratas
    top_idx = np.flip(np.argsort(predictions))[:N]

    # decode indeks untuk mendapatkan itemid yang sesuai
    top_items = self.iencoder.inverse_transform(top_idx)

    # ambil prediksi yang sesuai untuk indeks N teratas
    preds = predictions[top_idx]

    return top_items, preds
```

4. Melatih model *non matrix factorization* (NMF) dengan dataset ML-100k

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

    # membuat dan melatih 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))
```

```
Training NMF ...
k=10
epoch 1/10 - loss : 0.916 - val_loss : 0.917
epoch 2/10 - loss : 0.915 - val_loss : 0.917
epoch 3/10 - loss : 0.915 - val_loss : 0.917
epoch 4/10 - loss : 0.915 - val_loss : 0.917
epoch 5/10 - loss : 0.915 - val_loss : 0.917
epoch 6/10 - loss : 0.915 - val_loss : 0.917
epoch 7/10 - loss : 0.915 - val_loss : 0.917
epoch 8/10 - loss : 0.915 - val_loss : 0.917
epoch 9/10 - loss : 0.915 - val_loss : 0.917
epoch 10/10 - loss : 0.915 - val_loss : 0.917
```

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

```
validation error : 0.917
MAE :  0.9165041343019539
0.9165041343019539
```

Diperoleh MAE sekitar 91%.

5. Model NMF juga dapat dilakukan dengan Scikit-surprise sebagai berikut.

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

# load data
data = Dataset.load_builtin('ml-100k')

# menggunakan algoritma NMD
nmf = NMF(n_factors=10, n_epochs=10)

#Jalankan validasi silang 5 kali lipat dan cetak hasil.
history = cross_validate(nmf, data, measures=['MAE'], cv=5, verbose=True)
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting surprise
  Downloading surprise-0.1-py2.py3-none-any.whl (1.8 kB)
Collecting scikit-surprise
  Downloading scikit-surprise-1.1.3.tar.gz (771 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 772.0/772.0 KB 18.8 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-surprise->surprise) (1.2.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.8/dist-packages (from scikit-surprise->surprise) (1.21.6)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.8/dist-packages (from scikit-surprise->surprise) (1.7.3)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.3-cp38-cp38-linux_x86_64.whl size=2626477 sha256=a046f09447fc0d5d43c67
  Stored in directory: /root/.cache/pip/wheels/af/db/86/2c18183a80ba05da35bf0fb7417aac5cddbd93bcb1b92fd3ea
Successfully built scikit-surprise
Installing collected packages: scikit-surprise, surprise
Successfully installed scikit-surprise-1.1.3 surprise-0.1
Dataset ml-100k could not be found. Do you want to download it? [Y/n] Y
Trying to download dataset from https://files.grouplens.org/datasets/movielens/ml-100k.zip...
Done! Dataset ml-100k has been saved to /root/.surprise_data/ml-100k
Evaluating MAE of algorithm NMF on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
MAE (testset)     0.9626  0.9435  0.9777  0.9607  0.9510  0.9591  0.0116
```

7.7. Explainable Matrix Factorization

Metode ini akan mengkuantifikasi penjelasan dari suatu rekomendasi yang dibuat oleh model *collaborative* filtering. Langkah-langkah yang dilakukan adalah sebagai berikut.

1. Memuat *tools* yang diperlukan.

```
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
```

```
Saving to: 'recsys.zip'

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

2023-01-04 02:04:28 (105 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
```

```
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
```

2. Menghitung *explainable score* dengan *neighbourhood based similarities*.

```python
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
```

3. Membuat model *explainable matrix factorization* (EMF).

```python
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)

        # inisialisasi matriks faktor laten P dan Q (masing-masing bentuk (m,k) dan (n,k)) yang akan dipelajari
        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":[],
        }
```

```python
    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] - self.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] + self.lamb*(self.P[u] - self.Q[i]) * self.W[u,i])

    def mae(self,  x_train, y_train):
        """
        returns the Mean Absolute Error
        """
        # banyaknya training
        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))

        return self.history
```

```python
    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 known
            - 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()

        # mendapatkan 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)

            # kesalahan pelatihan dan validasi setelah epoch
            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_error, steps=1)

        return self.history
```

```python
    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
```

[4]

```python
    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
        """
        # menyandikan userid
        u = uencoder.transform([userid])[0]

        #prediksi untuk pengguna ini di semua produk
        predictions = np.dot(self.P[u], self.Q.T)

        # dapatkan indeks prediksi N teratas
        top_idx = np.flip(np.argsort(predictions))[:N]

        # decode indeks untuk mendapatkan itemid yang sesuai
        top_items = iencoder.inverse_transform(top_idx)

        # ambil prediksi yang sesuai untuk indeks N teratas
        preds = predictions[top_idx]

        return top_items, preds
```

```python
epochs = 10
```

4. Mengevaluasi model pada data Movielens 100k yang belum dinormalisasi.

```
[6] # memuat data
    ratings, movies = ml100k.load()

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

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

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

    # dapatkan contoh sebagai tupel userid dan itemid dan label dari normalisasi peringkat
    raw_examples, raw_labels = get_examples(ratings)

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

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

```
[7] # buat model pengguna ke pengguna untuk ukuran kesamaan
    usertouser = UserToUser(ratings, movies)

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

```
Normalize users ratings ...
Initialize the similarity model ...
Compute nearest neighbors ...
User to user recommendation model created with success ...
Compute Explainable score. Progress status : 99.9%
```

```
[9] # inisialisasi model
    EMF = ExplainableMatrixFactorization(m, n, W, alpha=0.01, beta=0.4, lamb=0.01, k=10)

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

```
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
```

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

```
validation error : 0.797
```

Diperoleh *validation error* 0.797.

5. Mengevaluasi model pada data Movielens 100k yang telah dinormalisasi.

```
# memuat 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())
```

```
[12] # inisialisasi model
EMF = ExplainableMatrixFactorization(m, n, W, alpha=0.022, beta=0.65, lamb=0.01, k=10)

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

```
Training EMF
k=10     alpha=0.022     beta=0.65       lambda=0.01
epoch 1/10 - loss : 0.809 - val_loss : 0.842
epoch 2/10 - loss : 0.809 - val_loss : 0.829
epoch 3/10 - loss : 0.807 - val_loss : 0.821
epoch 4/10 - loss : 0.799 - val_loss : 0.811
epoch 5/10 - loss : 0.789 - val_loss : 0.8
epoch 6/10 - loss : 0.782 - val_loss : 0.793
epoch 7/10 - loss : 0.778 - val_loss : 0.789
epoch 8/10 - loss : 0.776 - val_loss : 0.786
epoch 9/10 - loss : 0.774 - val_loss : 0.784
epoch 10/10 - loss : 0.773 - val_loss : 0.783
```

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

```
validation error : 0.783
```

Diperoleh *validation error* 0.783.

6. Mengevaluasi model pada data Movielens 1M yang belum dinormalisasi.

```
[14] # memuat 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)

# dapatkan contoh sebagai tupel userid dan itemid dan label dari normalisasi peringkat
raw_examples, raw_labels = get_examples(ratings)

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

```
Download data 100.1%
Successfully downloaded ml-1m.zip 5917549 bytes.
Unzipping the ml-1m.zip zip file ...
```

```
[15] # buat model pengguna ke pengguna untuk ukuran kesamaan
     usertouser = UserToUser(ratings, movies)

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

```
Normalize users ratings ...
Initialize the similarity model ...
Compute nearest neighbors ...
User to user recommendation model created with success ...
Compute Explainable score. Progress status : 100.0%
```

```
[16] # inisialisasi model
     EMF = ExplainableMatrixFactorization(m, n, W, alpha=0.01, beta=0.4, lamb=0.01, k=10)

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

```
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
```

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

```
validation error : 0.76
```

Diperoleh *validation error* 0.76.

7. Mengevaluasi model pada data Movielens 1M yang telah dinormalisasi.

```
[18] # memuat data
     ratings, movies = ml1m.load()

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

     # menormalkan peringkat dengan cara mengurangi
     normalized_column_name = "norm_rating"
     ratings = normalized_ratings(ratings, norm_column=normalized_column_name)

     # dapatkan contoh sebagai tupel userid dan itemid dan label dari normalisasi peringkat
     raw_examples, raw_labels = get_examples(ratings, labels_column=normalized_column_name)

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

```
[19] # inisialisasi model
     EMF = ExplainableMatrixFactorization(m, n, W, alpha=0.023, beta=0.59, lamb=0.01, k=10)

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

```
Training EMF
k=10     alpha=0.023     beta=0.59       lambda=0.01
epoch 1/10 - loss : 0.805 - val_loss : 0.814
epoch 2/10 - loss : 0.764 - val_loss : 0.77
epoch 3/10 - loss : 0.756 - val_loss : 0.762
epoch 4/10 - loss : 0.755 - val_loss : 0.759
epoch 5/10 - loss : 0.754 - val_loss : 0.759
epoch 6/10 - loss : 0.754 - val_loss : 0.758
epoch 7/10 - loss : 0.754 - val_loss : 0.758
epoch 8/10 - loss : 0.753 - val_loss : 0.758
epoch 9/10 - loss : 0.753 - val_loss : 0.758
epoch 10/10 - loss : 0.753 - val_loss : 0.758
```

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

```
validation error : 0.758
```

Diperoleh *validation error* 0.758.

7.8. Performance Measure

Dalam tahap ini, akan dibandingkan performa dari masing-masing model yang dipaparkan di atas. Langkah-langkah yang dilakukan adalah sebagai berikut.

1. Memuat *tools* yang diperlukan.

```
[1] 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
```

```
--2023-01-04 03:06:43--  https://github.com/nzhinusoftcm/review-on-collaborative-filtering/raw/master/recsys.zip
Resolving github.com (github.com)... 140.82.112.4
Connecting to github.com (github.com)|140.82.112.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/nzhinusoftcm/review-on-collaborative-filtering/master/recsys.zip [following]
--2023-01-04 03:06:43--  https://raw.githubusercontent.com/nzhinusoftcm/review-on-collaborative-filtering/master/recsys.zip
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.133, 185.199.111.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109.133|: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

2023-01-04 03:06:44 (134 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
```

```python
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
```

2. Membandingkan hasil dengan data Movielens 100k
   a. Hasil dari user-based collaborative filtering dengan jarak euclidean dan

```python
# 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 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(examples=raw_examples, labels=raw_labels)

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

```
Download data 100.2%
Successfully downloaded ml-100k.zip 4924029 bytes.
Unzipping the ml-100k.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 10000 test data ...

MAE : 0.8125945111976461
0.8125945111976461
```

```
[4] # evaluate with cosine similarity
    usertouser = UserToUser(ratings, movies, metric='cosine')
    print("==========================")
    usertouser.evaluate(x_test, y_test)
```

```
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. Hasil dari item-based collaborative filtering dengan jarak euclidean dan

```
# 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 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(examples=raw_examples, labels=raw_labels)

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

```
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
```

```
[6] # evaluation with Euclidean distance
    itemtoitem = ItemToItem(ratings, movies, metric='euclidean')
    print("==================")
    itemtoitem.evaluate(x_test, y_test)
```

```
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. Hasil dari matrix factorization dengan 10 epochs.

```
[7] epochs = 10
```

```
[8] # 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(examples=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_data=(x_test, y_test))
```

```
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. Hasil dari non-negative matrix factorization untuk 10 epochs.

```
!pip install 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=True)
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting surprise
  Downloading surprise-0.1-py2.py3-none-any.whl (1.8 kB)
Collecting scikit-surprise
  Downloading scikit-surprise-1.1.3.tar.gz (771 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 772.0/772.0 KB 16.8 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-surprise->surprise) (1.2.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.8/dist-packages (from scikit-surprise->surprise) (1.21.6)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.8/dist-packages (from scikit-surprise->surprise) (1.7.3)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.3-cp38-cp38-linux_x86_64.whl size=2626477 sha256=3826150b276760f50778777daf880d453c0d6b20c0
  Stored in directory: /root/.cache/pip/wheels/af/db/86/2c18183a80ba05da35bf0fb7417aac5cddbd93bcb1b92fd3ea
Successfully built scikit-surprise
Installing collected packages: scikit-surprise, surprise
Successfully installed scikit-surprise-1.1.3 surprise-0.1
Dataset ml-100k could not be found. Do you want to download it? [Y/n] Y
Trying to download dataset from https://files.grouplens.org/datasets/movielens/ml-100k.zip...
Done! Dataset ml-100k has been saved to /root/.surprise_data/ml-100k
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          0.56    0.60    0.59    0.44    0.44    0.52    0.07
Test time         0.14    0.38    0.23    0.27    0.13    0.23    0.09
```

e. Hasil dari explainable matrix factorization.

```
[11] # 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(examples=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_data=(x_test, y_test))

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

```
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
0.797347824723284
```

3. Membandingkan hasil dengan data Movielens 1M

a. Hasil dari user-based collaborative filtering dengan jarak euclidean dan cosine.

```python
# load ml1M 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(examples=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)
```

```
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
```

```python
# metric : euclidean

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

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

```
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 : 0.8069332535426614
0.8069332535426614
```

b. Hasil dari item-based collaborative filtering dengan jarak euclidean dan cosine.

Cosine similarity

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

```
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

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

```
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. Hasil dari *matrix factorization*

```
[17] # 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(examples=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)
```

```
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. Hasil dari *non negative matrix factorization*.

```
[19] 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, verbose=True)
```

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
MAE (testset)     0.9509  0.9708  0.9527  0.9670  0.9542  0.9591  0.0081
Fit time          4.90    6.51    5.67    6.03    6.21    5.86    0.55
Test time         2.58    3.17    3.15    3.73    2.96    3.12    0.37
```

e. Hasil dari *explainable matrix factorization*.

```python
# 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(examples=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_data=(x_test, y_test))
```

```
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
```

Ringkasan perbandingan performa dapat disajikan dalam tabel berikut.

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

Dari perbandingan tersebut, dapat diketahui bahwa MAE dipengaruhi oleh jarak yang digunakan (untuk *user based* dan *item based collaborative filtering*), serta proses normalisasi data. Untuk jarak euclidean, MAE *user based collaborative filtering* lebih rendah dibanding *item based*, sedangkan untuk jarak cosine *item based colaborative filtering* lebih rendah dibanding *user based*. Untuk model MF, NMF, maupun EMF, proses normalisasi data dapat berpengaruh terhadap performa di mana MAE cenderung menurun.

KESIMPULAN

1. *Mean relative error* (MAE) dipengaruhi oleh jarak yang digunakan (untuk *user based* dan *item based collaborative filtering*), serta proses normalisasi data.
2. Untuk jarak euclidean, MAE *user based collaborative filtering* lebih rendah dibanding *item based*, sedangkan untuk jarak cosine *item based colaborative filtering* lebih rendah dibanding *user based*.
3. Untuk model MF, NMF, maupun EMF, proses normalisasi data dapat berpengaruh terhadap performa di mana MAE cenderung menurun.

DAFTAR PUSTAKA

Abdollahi and Nasraoui (2016). *Explainable Matrix Factorization for Collaborative Filtering*

Abdollahi and Nasraoui (2017). *Using Explainability for Constrained Matrix Factorization*

Daniel Billsus and Michael J. Pazzani (1998). *Learning Collaborative Information Filters*

Daniel D. Lee & H. Sebastian Seung (1999). *Learning the parts of objects by non-negative matrix factorization*

Deng Cai et al. (2008). *Non-negative Matrix Factorization on Manifold*

Herlocker et al. (1999) *An Algorithmic Framework for Performing Collaborative Filtering*

Sarwar et al. (2001) *Item-based collaborative filtering recommendation algorithms*

Michael D. Ekstrand, et al. (2011). *Collaborative Filtering Recommender Systems*

George Karypis (2001) *Evaluation of Item-Based Top-N Recommendation Algorithms*

Greg Linden, Brent Smith, and Jeremy York (2003) Amazon.com *Recommendations : Item-to-Item Collaborative Filtering*

Mason, n.d. *Short History of Collaborative Filtering.* Diakses dari https://www.moyak.com/papers/collaborative-filtering.html tanggal 4 Januari 2023.

Sarwar et al. (2001) *Item-based collaborative filtering recommendation algorithms*

Willmott, Cort J.; Matsuura, Kenji (December 19, 2005). *Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance*. *Climate Research*. **30**: 79–82

Sarwar et al. (2000). *Application of Dimensionality Reduction in Recommender System -- A Case Study*

Yu-Xiong Wang and Yu-Jin Zhang (2011). *Non-negative Matrix Factorization: a Comprehensive Review*

Yehuda Koren et al. (2009). *Matrix Factorization Techniques for Recommender Systems*

Shuo Wang et al, (2018). *Explainable Matrix Factorization with Constraints on Neighborhood in the Latent Space*

Terveen, Loren; Hill, Will (2001). *Beyond Recommender Systems: Helping People Help Each Other*. Addison-Wesley. p. 6. Retrieved 16 January 2012.