Laporan Tugas Besar MK Recommender System:
Hybrid Recommender System

Nama : Shindy Trimaria Laxmi
NIM : 1301170092
Kelas : ICM-40-GAB
Youtube : https://youtu.be/x5e_zedhRPU

SUMMARY

Pada tugas besar ini, telah dibangun sebuah hybrid recommender system untuk memberikan rekomandasi film terhadap user. Data yang digunakan merupakan data MovieLens, lalu menggunakan tiga file untuk membangun sistem ini, yaitu:

1. Movies.csv : movieId, title, genre
2. Ratings.csv : userId, movieId, rating, timestamp
3. Tags.csv : userId, movieId, tag, timestamp

Pada tugas besar ini akan digunakan pipelined hybridization design dengan mengkombinasikan dua teknik rekomendasi. yaitu Collaborative Filtering (ratings.csv) dan Content Based Filtering (movies.csv dan tags.csv).

COLLABORATIVE FILTERING

Pada sistem ini, dilakukan collaborative filtering (CF) dengan menggunakan file ratings.csv. CF itu sendiri berfungsi untuk mengatasi masalah sparse dan digunakan untuk memprediksi rating yang akan diberikan oleh suatu user terhadap suatu item. Pada CF, dilakukan 5-fold validations terlebih dahulu untuk mencari kemungkinan kombinasi data training dan data testing yang memiliki akurasi paling bagus.

Pada sistem ini 5-fold disiapkan secara manual dengan cara melakukan split data train dan data testing sebanyak 5 kali. Data testing yang digunakan berukuran 20% dari seluruh data. Lalu di copy data testing ke variabel temp, yang nantinya pada temp, column rating akan diubah menjadi 0. Setelah diubah menjadi, data training akan di gabungkan dengan data temp agar dimensi matrix data train tetap sama dengan dimensi data aslinya.

```
# 5-FOLD VALIDATIONS
rating_train1, rating_test1 = train_test_split(ratings, test_size = 0.2 , shuffle=True)
temp1 = copy.deepcopy(rating_test1)
temp1.loc[:,'rating'] = 0
rating_train1_df = pd.concat([rating_train1, temp1])

rating_train2, rating_test2 = train_test_split(ratings, test_size = 0.2 , shuffle=True)
temp2 = copy.deepcopy(rating_test2)
temp2.loc[:,'rating'] = 0
rating_train2_df = pd.concat([rating_train2, temp2])

rating_train3, rating_test3 = train_test_split(ratings, test_size = 0.2 , shuffle=True)
temp3 = copy.deepcopy(rating_test3)
temp3.loc[:,'rating'] = 0
rating_train3_df = pd.concat([rating_train3, temp3])

rating_train4, rating_test4 = train_test_split(ratings, test_size = 0.2 , shuffle=True)
temp4 = copy.deepcopy(rating_test4)
temp4.loc[:,'rating'] = 0
rating_train4_df = pd.concat([rating_train4, temp4])

rating_train5, rating_test5 = train_test_split(ratings, test_size = 0.2 , shuffle=True)
temp5 = copy.deepcopy(rating_test5)
temp5.loc[:,'rating'] = 0
rating_train5_df = pd.concat([rating_train5, temp5])
```

Lalu data train akan diubah ke bentuk DataFrame, yang nantinya data tersebut akan digunakan untuk melakukan CF.

```
rating2 = rating_train2_df.pivot(index='userId', columns='movieId', values='rating')
rating2 = rating2.fillna(0) # Rating kosong diganti dengan 0
rating2
```

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573	193579	193581	193583	193585	193587	193609
userId																					
1	4.0	0.0	4.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
606	2.5	0.0	0.0	0.0	0.0	0.0	2.5	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
607	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
608	2.5	2.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
609	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
610	5.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

610 rows × 9724 columns

Kemudian setiap kombinasi data train akan diimplementasikan algoritma Matrix Factorization untuk mencari prediksi rating yang akan diberikan oleh user terhadap item.

```python
def matrix_factorization(R, P, Q, K, steps=5, alpha=0.02, beta=0.2):
    Q = Q.T
    for step in range(steps):
        for i in range(len(R)):
            for j in range(len(R[i])):
                if R[i][j] > 0:
                    eij = R[i][j] - np.dot(P[i,:],Q[:,j])
                    for k in range(K):
                        P[i][k] = P[i][k] + alpha * (2 * eij * Q[k][j] - beta * P[i][k])
                        Q[k][j] = Q[k][j] + alpha * (2 * eij * P[i][k] - beta * Q[k][j])
        eR = np.dot(P,Q)
        e = 0
        for i in range(len(R)):
            for j in range(len(R[i])):
                if R[i][j] > 0:
                    e = e + pow(R[i][j] - np.dot(P[i,:],Q[:,j]), 2)
                    for k in range(K):
                        e = e + (beta/2) * ( pow(P[i][k],2) + pow(Q[k][j],2) )
        if e < 0.001:
            break
    return P, Q.T
```

```python
def getRatingPredictions(data):
    R = data.to_numpy()

    N = len(R)
    M = len(R[0])
    K = 2

    P = np.random.rand(N,K)
    Q = np.random.rand(M,K)

    user_latent_features, item_latent_features = matrix_factorization(R, P, Q, K)
    return np.dot(user_latent_features, item_latent_features.T)
```

```python
pred1 = getRatingPredictions(rating1)
print('Fold 1 DONE')
pred2 = getRatingPredictions(rating2)
print('Fold 2 DONE')
pred3 = getRatingPredictions(rating3)
print('Fold 3 DONE')
pred4 = getRatingPredictions(rating4)
print('Fold 4 DONE')
pred5 = getRatingPredictions(rating5)
print('Fold 5 DONE')
```

```
Fold 1 DONE
Fold 2 DONE
Fold 3 DONE
Fold 4 DONE
Fold 5 DONE
```

Setelah didapatkan prediksi rating berdasarkan data training, hasil dari prediksi tersebut akan dievaluasi dengan menggunakan recall. Kondisi yang diberlakukan pada saat evaluasi adalah menghitung akurasi item yang diberi rating lebih dari 4 oleh user. Dan dari setiap hasil prediksi rating didaptkan hasil sebagai berikut.

```
[0.5902793549852373,
 0.5880576208178439,
 0.5883152173913043,
 0.5855092919313103,
 0.5895504824664627]
```

Dapat terlihat bahwa kombinasi data train dan data tes pada fold ke 1 memiliki akurasi paling baik, sehingga untuk kedepannya akan digunakan hasil prediksi dari data train fold ke 1.

```
print("THE ORIGINAL MATRIX")
rating
```

```
THE ORIGINAL MATRIX
```

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573	193579	193581	193583	193585	193587	193609
userId																					
1	4.0	0.0	4.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
606	2.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
607	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
608	2.5	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
609	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
610	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

610 rows × 9724 columns

```
print("THE PREDICTED MATRIX USING MATRIX FACTORIZATION")
predict
```

```
THE APPROXIMATION MATRIX BY MF
```

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573	19
userId																
1	3.754333	4.311571	3.461073	2.946872	2.990490	4.381082	3.599101	3.488525	3.428278	4.023413	...	2.827272	2.610737	1.844916	1.694750	3.5
2	3.292882	3.785213	3.038544	2.586122	2.637244	3.858966	3.143117	3.068279	2.995737	3.548939	...	2.494581	2.292234	1.641451	1.490101	3.1
3	2.894937	3.398591	2.728194	2.302331	2.601575	3.716151	2.494196	2.866155	2.412995	3.516352	...	2.485909	2.062390	1.903538	1.382272	2.9
4	2.731599	3.119852	2.504430	2.137127	2.107153	3.109100	2.683952	2.497269	2.547925	2.831208	...	1.986030	1.888088	1.230596	1.215543	2.5
5	2.982484	3.406089	2.734204	2.333288	2.299458	3.393254	2.931627	2.725900	2.782897	3.089524	...	2.167169	2.061296	1.341626	1.326872	2.8
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
606	2.996924	3.449005	2.768657	2.355309	2.416201	3.530404	2.845421	2.802035	2.714019	3.252352	...	2.286911	2.088877	1.519929	1.360255	2.8
607	2.933402	3.377183	2.711003	2.305907	2.370109	3.461429	2.780243	2.745697	2.652501	3.190587	...	2.243736	2.045456	1.496044	1.332730	2.8
608	3.595693	4.140309	3.323595	2.826785	2.907784	4.245863	3.405518	3.367134	3.249373	3.914530	...	2.752967	2.507696	1.837982	1.634282	3.4
609	2.798821	3.204975	2.572763	2.193107	2.192367	3.223744	2.718287	2.578604	2.584614	2.947589	...	2.069413	1.940112	1.315150	1.253972	2.6
610	3.858923	4.437118	3.561855	3.031172	3.095514	4.527953	3.678725	3.598650	3.506853	4.165901	...	2.928497	2.687087	1.931637	1.747506	3.6

CONTENT BASED FILTERING

Content Based Filtering akan diimplementasikan untuk melakukan :

1. Clustering berdasarkan genre dengan menggunakan file movies.csv, dan
2. Clustering berdasarkan tag dengan menggunakan file tags.csv .

Teknik clustering yang digunakan adalah algoritma TFIDF dan mengelompokkan berdasarkan cosine similarity antara setiap movie ke movie lainnya.

Clustering Berdasarkan Genre Movie

Data yang digunakan pada proses ini adalah file movies.csv. Sebelum di implementasikan algoritma TFIDF, data akan di preprocessing terlebih dahulu. Tahapan preprocessing yang dilakukan adalah:

1. Mengubah semua data pada kolom genre menjadi lowercase
2. Menghapus spasi dan karakter (-) pada data kolom genre

Preprocessing dilakukan agar pada saat diimplementasikan algoritma TFIDF, features yang didapatkan lebih tepat dan bukan features yang terpisah.

```
movies = pd.read_csv('Data/movies.csv', sep=',')
movies.genres = movies.genres.str.lower()
movies.genres = movies.genres.str.replace(" ", "")
movies.genres = movies.genres.str.replace("-", "")
movies
```

	movieId	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

Lalu akan diimplementasikan algoritma TFIDF terhadap kolom genres karena movie akan di kelompokkan berdasarkan genre.

```
v = TfidfVectorizer()
tfidf = v.fit_transform(movies['genres'])
tfidf.shape
```

```
(9742, 20)
```

Setelah didapatkan vektor TFIDF dari setiap movie pada file movies. Selanjutnya akan dicari cosine similarity dari setiap movie tersebut.

```
cosine_similarity = pd.DataFrame(data=cos_sim, index=movies.movieId, columns=movies.movieId)
cosine_similarity
```

movieId	1	2	3	4	5	6	7	8	9	10	...	193565	193567	193571	193573
movieId															
1	1.000000	0.813578	0.152769	0.135135	0.267586	0.000000	0.152769	0.654698	0.000000	0.262413	...	0.411168	0.465621	0.196578	0.516225
2	0.813578	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.804715	0.000000	0.322542	...	0.000000	0.000000	0.000000	0.000000
3	0.152769	0.000000	1.000000	0.884571	0.570915	0.000000	1.000000	0.000000	0.000000	0.000000	...	0.185790	0.000000	0.419413	0.000000
4	0.135135	0.000000	0.884571	1.000000	0.505015	0.000000	0.884571	0.000000	0.000000	0.000000	...	0.164344	0.201391	0.687440	0.000000
5	0.267586	0.000000	0.570915	0.505015	1.000000	0.000000	0.570915	0.000000	0.000000	0.000000	...	0.325424	0.000000	0.734632	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
193581	0.680258	0.341376	0.181883	0.160888	0.318581	0.239513	0.181883	0.000000	0.436010	0.241142	...	0.683714	0.554355	0.234040	0.614603
193583	0.755891	0.379331	0.202105	0.178776	0.354002	0.000000	0.202105	0.000000	0.000000	0.000000	...	0.543952	0.615990	0.260061	0.682937
193585	0.000000	0.000000	0.000000	0.466405	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.431794	0.678466	0.000000
193587	0.421037	0.000000	0.000000	0.000000	0.000000	0.317844	0.000000	0.000000	0.578606	0.320007	...	0.769740	0.735655	0.000000	0.815607
193609	0.267586	0.000000	0.570915	0.505015	1.000000	0.000000	0.570915	0.000000	0.000000	0.000000	...	0.325424	0.000000	0.734632	0.000000

9742 rows × 9742 columns

Lalu dilakukan pencarian 20 movie yang paling mirip dengan suatu movie dan ditampung kedalam variable genre. Salah satu contoh hasil pengelompokkan tersebut adalah sebagai berikut:

```
genre[0]
```

```
[1,
 [2294,
  3114,
  3754,
  4016,
  4886,
  45074,
  53121,
  65577,
  91355,
  103755,
  136016,
  166461,
  134853,
  2033,
  2116,
  3400,
  4366,
  4519,
  5672,
  6536]]
```

Terlihat bahwa movie dengan id 1 memiliki 20 movie yang sama pada list yang terdapat pada data tersebut.

Clustering Berdasarkan Tag Movie

Data yang digunakan pada proses ini adalah file tags.csv. Sebelum di implementasikan algoritma TFIDF, data akan di preprocessing terlebih dahulu. Tahapan preprocessing yang dilakukan adalah:

1. Mengubah semua data pada kolom genre menjadi lowercase
2. Menghapus spasi dan karakter karakter spesial pada data kolom genre

Preprocessing dilakukan agar pada saat diimplementasikan algoritma TFIDF, features yang didapatkan lebih tepat dan bukan features yang terpisah.

```
tags = pd.read_csv('Data/tags.csv', sep=',')
tags.tag = tags.tag.str.lower()
tags.tag = tags.tag.str.replace(" ","")
tags.tag = tags.tag.str.replace("'","")

def strip_character(data):
    r = re.compile(r'[^a-z !@#$%&*_+-=|\:;<>,./()[\]{}\"]')
    return r.sub('', data)

tags.tag = tags.tag.apply(strip_character)

tags
```

	userId	movieId	tag	timestamp
0	2	60756	funny	1445714994
1	2	60756	highlyquotable	1445714996
2	2	60756	willferrell	1445714992
3	2	89774	boxingstory	1445715207
4	2	89774	mma	1445715200

Kemudian data akan di grouping berdasarkan movieId, untuk mengetahui tag apa saja yang dimiliki oleh movie tersebut. Sebelum digrouping, akan dilakukan penghapusan kolom timestamp karena tidak digunakan pada proses ini.

```
tags = tags.drop(['timestamp'],axis=1)
tags = tags.groupby(['movieId'], as_index=False)['userId','tag'].agg(lambda x: list(x))
tags
```

	movieId	userId	tag
0	1	[336, 474, 567]	[pixar, pixar, fun]
1	2	[62, 62, 62, 474]	[fantasy, magicboardgame, robinwilliams, game]
2	3	[289, 289]	[moldy, old]
3	5	[474, 474]	[pregnancy, remake]
4	7	[474]	[remake]

Karena masih terdapat duplikat pada kolom tag dan userId, maka akan dilakukan penghapusan duplikat.

```
def removeDuplicate(data):
    temp = ""
    for i in data:
        if str(i) not in temp:
                temp += str(i) + "|"
    return temp[:len(temp)-1]

tags.userId = tags.userId.apply(removeDuplicate)
tags.tag = tags.tag.apply(removeDuplicate)
tags
```

	movieId	userId	tag
0	1	336|474|567	pixar|fun
1	2	62|474	fantasy|magicboardgame|robinwilliams
2	3	289	moldy
3	5	474	pregnancy|remake
4	7	474	remake

Lalu akan diimplementasikan algoritma TFIDF terhadap kolom tag karena movie akan di kelompokkan berdasarkan tag.

```
v2 = TfidfVectorizer()
tfidf2 = v2.fit_transform(tags.tag)
tfidf2.shape
```

```
(1572, 1495)
```

Setelah didapatkan vektor TFIDF dari setiap movie pada file tags. Selanjutnya akan dicari cosine similarity dari setiap movie tersebut.

```
cosine_similarity2 = pd.DataFrame(data=cos_sim2, index=tags.movieId, columns=tags.movieId)
cosine_similarity2
```

movieId	1	2	3	5	7	11	14	16	17	21	...	176371	176419	179401	180031	180985	183611	184471	187593	187595
movieId																				
1	1.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0
2	0.0	1.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0
3	0.0	0.0	1.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0
5	0.0	0.0	0.0	1.000000	0.680832	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0
7	0.0	0.0	0.0	0.680832	1.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
183611	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.437843	0.0	0.0	1.000000	0.0	0.0	0.0
184471	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.000000	1.0	0.0	0.0
187593	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	1.0	0.0
187595	0.0	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	1.0

Lalu dilakukan pencarian 20 movie yang paling mirip dengan suatu movie dan ditampung kedalam variable tag. Salah satu contoh hasil pengelompokkan tersebut adalah sebagai berikut:

```
tag[0]
```

```
[1,
 [2355,
  122918,
  3114,
  89745,
  108932,
  68954,
  296,
  2,
  3,
  5,
  7,
  11,
  14,
  16,
  17,
  21,
  22,
  25,
  26,
  28]]
```

Terlihat bahwa movie dengan id 1 memiliki 20 movie yang sama pada list yang terdapat pada data tersebut.

HYBRIDIZATION STEP

Pada proses hybridization kita akaan melakukan rekomendasi movie terhadap user berdasarkan variable genre dari proses CBF genre movie, tag dari proses CBF tag movie dan prediksi rating dari proses CF. Kita kan menggunakan pipelined hybridization design. Tahapannya adalah sebagai berikut:

1. Mencari daftar movie yang telah ditonton oleh user. Daftar movie berdasarkan pada data training yang digunakan pada proses CF. Lalu dilakukan grouping berdasarkan userId dan kemudian melakukan penghapusan kolom rating dan timestamp.

```
watched = rating_train.groupby(['userId'], as_index=False)['movieId','rating','timestamp'].agg(lambda x: list(x))
watched = watched.drop(['rating','timestamp'],axis=1)
watched
```

	userId	movieId
0	1	[163, 2478, 1580, 3053, 2046, 2985, 593, 1031,...
1	2	[86345, 74458, 58559, 1704, 318, 89774, 46970,...
2	3	[6238, 5919, 647, 1371, 2080, 72378, 2288, 130...
3	4	[2599, 45, 1304, 593, 3033, 1196, 3408, 902, 3...
4	5	[357, 590, 410, 349, 608, 318, 531, 150, 515, ...

2. Kemudian akan dicari judul dari setiap movie id, karen proses rekomendasi akan menggunakan judul film, bukan movie id.

```
movie_title = pd.DataFrame(index=movies.movieId, columns = ['Title'], data=np.array([movies.title]).T)
movie_title
```

movieId	Title
1	Toy Story (1995)
2	Jumanji (1995)
3	Grumpier Old Men (1995)
4	Waiting to Exhale (1995)
5	Father of the Bride Part II (1995)

3. Disediakan beberapa function yang akan digunakan untuk menunjang proses rekomendasi movie. Fungsi dari tiap function tersebut adalah:
   a. GetMovieCandidate : untuk mencari semua movie yang mirip terhadap suatu movie berdasarkan genre dan tag dari movie tersebut.
   b. GetMovieList : mencari semua movie yang mirip terhadap semua movie yang telah di tonton oleh user.
   c. GetRatingList : untuk mencari rating dari semua list movie yang dihasilkan oleh getMovieList

```
def getMovieCandidate(movie_id, movie):
    dataG = list(filter(lambda x: x[0]==movie_id, genre))
    dataG = list(map(list, np.array(dataG)[:,1]))
    movie += [int(i) for i in ((str(dataG)[2:-2]).replace(",","")).split()]

    dataT = list(filter(lambda x: x[0]==movie_id, tag))
    dataT = list(map(list, np.array(dataT)[:,1]))
    movie += [int(i) for i in ((str(dataT)[2:-2]).replace(",","")).split()]

    movie = list(OrderedDict.fromkeys(movie))
    return movie

def getMovieList(movieId):
    movie = []
    for id in movieId:
        movie = getMovieCandidate(id,movie)
    return movie

def getRatingList(movie):
    rate = []
    for i in movie:
        if i in predict.columns:
            rate.append([i,predict[i].loc[user.userId]])
    return rate
```

d. GetRecommendation : untuk mencari rekomendasi movie terhadap suatu user berdasarkan rating yang dihasilkan oleh getRatingList. Untuk mendapatkan rekomendasi film, list movie yang mirip akan di sorting secara descendein berdasarkan rating. Kemudian akan diambil 20 movie teratas.

```
def getRecommendation(user):
    rec = []
    movie = getMovieList(user.movieId)
    rate = getRatingList(movie)
    rate.sort(key = lambda rate: rate[1], reverse=True)
    rate = rate[:20]
    for i in rate:
        rec.append(movie_title.Title.loc[i[0]])
    return rec
```

4. Kemudian akan dilakukan pencarian rekomendasi movie terhadap setiap user yang ada pada data training.

```
rec_list = []
print('Get movie recommendations for every user')
for i in range(len(watched)):
    user = watched.loc[i]
    rec_list.append(getRecommendation(user))
    print('User',user.userId,'done.')
```

5. Sehingga didapatkan hasil rekomendasi seperti berikut:

```
movie_recommendations = pd.DataFrame(data=rec_list, index=watched.userId, columns=['Movie1','Movie2','Movie3','Movie4','Movie5'
movie_recommendations
```

userId	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movie9	Movie10	Movie11	Movie12	Movie13
1	Yojimbo (1961)	Lord of the Rings: The Fellowship of the Ring,...	Matrix, The (1999)	Fight Club (1999)	Kolya (Kolja) (1996)	L.A. Confidential (1997)	Life Is Beautiful (La Vita è bella) (1997)	Seven Samurai (Shichinin no samurai) (1954)	Verdict, The (1982)	Man Bites Dog (C'est arrivé près de chez vous)...	Philadelphia Story, The (1940)	Wings of Desire (Himmel über Berlin, Der) (1987)	Reservoi Dogs (1992
2	Matrix, The (1999)	Fight Club (1999)	In the Mood For Love (Fa yeung nin wa) (2000)	Life Is Beautiful (La Vita è bella) (1997)	Seven Samurai (Shichinin no samurai) (1954)	Man Bites Dog (C'est arrivé près de chez vous)...	Reservoir Dogs (1992)	400 Blows, The (Les quatre cents coups) (1959)	Crouching Tiger, Hidden Dragon (Wo hu cang lon...	Schindler's List (1993)	Rear Window (1954)	Godfather: Part II, The (1974)	Dr Strangelove or: How Learned to Stop Worr..