Building Collaborative Filters

Juan David Serna Valderrama

In [1]:
```python
import pandas as pd
import numpy as np
```

In [2]:
```python
u_cols = ['user_id', 'age', 'sex', 'occupation', 'zip_code']

users = pd.read_csv("C:/Users/juand/OneDrive/Escritorio/Recommendation Systems with Python/Data/movi/u.user.csv",
                    sep='|', names =u_cols, encoding='latin-1')

users.head()
```

Out[2]:

	user_id	age	sex	occupation	zip_code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213

In [4]:
```python
i_cols = ['movie_id', 'title' ,'release date','video release date', 'IMDb URL', 'unknown', 'Action', 'Adventure',
 'Animation', 'Children\'s', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy',
 'Film-Noir', 'Horror', 'Musical', 'Mystery', 'Romance', 'Sci-Fi', 'Thriller', 'War', 'Western']

movies = pd.read_csv("C:/Users/juand/OneDrive/Escritorio/Recommendation Systems with Python/Data/movi/u.item.csv",
                     sep='|', names=i_cols, encoding='latin-1')

movies.head()
```

Out[4]:

	movie_id	title	release date	video release date	IMDb URL	unknown	Action	Adventure	Animation	Children's	...	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War	We
0	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	0	0	0	1	1	...	0	0	0	0	0	0	0	0	0	
1	2	GoldenEye (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?GoldenEye%20(...	0	1	1	0	0	...	0	0	0	0	0	0	0	1	0	
2	3	Four Rooms (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Four%20Rooms%...	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0	
3	4	Get Shorty (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Get%20Shorty%...	0	1	0	0	0	...	0	0	0	0	0	0	0	0	0	
4	5	Copycat (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Copycat%20(1995)	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0	

5 rows × 24 columns

In [5]:
```python
#Remove all information except Movie ID and title
movies = movies[['movie_id', 'title']]
```

In [6]:
```python
#Load the u.data file into a dataframe
r_cols = ['user_id', 'movie_id', 'rating', 'timestamp']

ratings = pd.read_csv("C:/Users/juand/OneDrive/Escritorio/Recommendation Systems with Python/Data/movi/u.data.csv",
                      sep='\t', names=r_cols,
 encoding='latin-1')

ratings.head()
```

Out[6]:

	user_id	movie_id	rating	timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

In [7]:
```python
#Drop the timestamp column
ratings = ratings.drop('timestamp', axis=1)
```

In [8]:
```python
#Import the train_test_split function
from sklearn.model_selection import train_test_split

#Assign X as the original ratings dataframe and y as the user_id column of ratings.
X = ratings.copy()
y = ratings['user_id']

#Split into training and test datasets, stratified along user_id
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, stratify=y, random_state=42)
```

In [9]:
```python
#Import the mean_squared_error function
from sklearn.metrics import mean_squared_error

#Function that computes the root mean squared error (or RMSE)
def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))
```

In [10]:
```python
#Define the baseline model to always return 3.
def baseline(user_id, movie_id):
    return 3.0
```

In [11]:
```python
#Function to compute the RMSE score obtained on the testing set by a model
def score(cf_model):

    #Construct a list of user-movie tuples from the testing dataset
    id_pairs = zip(X_test['user_id'], X_test['movie_id'])

    #Predict the rating for every user-movie tuple
    y_pred = np.array([cf_model(user, movie) for (user, movie) in id_pairs])

    #Extract the actual ratings given by the users in the test data
    y_true = np.array(X_test['rating'])

    #Return the final RMSE score
    return rmse(y_true, y_pred)
```

In [12]:
```python
score(baseline)
```

Out[12]:
```
1.2470926188539486
```

User Based Collaborative Filtering

Ratings Matrix

In [13]:
```python
#Build the ratings matrix using pivot_table function
r_matrix = X_train.pivot_table(values='rating', index='user_id', columns='movie_id')

r_matrix.head()
```

Out[13]:

movie_id	1	2	3	4	5	6	7	8	9	10	...	1669	1670	1671	1673	1674	1675	1676	1679	1681	1682
user_id																					
1	5.0	3.0	4.0	3.0	3.0	5.0	4.0	1.0	5.0	3.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	...	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	NaN	3.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 1647 columns

Mean

In [14]:
```python
#User Based Collaborative Filter using Mean Ratings
def cf_user_mean(user_id, movie_id):

    #Check if movie_id exists in r_matrix
    if movie_id in r_matrix:
        #Compute the mean of all the ratings given to the movie
        mean_rating = r_matrix[movie_id].mean()

    else:
        #Default to a rating of 3.0 in the absence of any information
        mean_rating = 3.0

    return mean_rating
```

In [15]:
```python
#Compute RMSE for the Mean model
score(cf_user_mean)
```

Out[15]:
```
1.0234701463131335
```

Weighted Mean

In [16]:
```python
#Create a dummy ratings matrix with all null values imputed to 0
r_matrix_dummy = r_matrix.copy().fillna(0)
```

In [17]:
```python
# Import cosine_score
from sklearn.metrics.pairwise import cosine_similarity

#Compute the cosine similarity matrix using the dummy ratings matrix
cosine_sim = cosine_similarity(r_matrix_dummy, r_matrix_dummy)
```

In [18]:
```python
#Convert into pandas dataframe
cosine_sim = pd.DataFrame(cosine_sim, index=r_matrix.index, columns=r_matrix.index)

cosine_sim.head(10)
```

Out[18]:

user_id	1	2	3	4	5	6	7	8	9	10	...	934	935	936	937	938	939	940	941	
user_id																				
1	1.000000	0.118076	0.029097	0.011628	0.264677	0.312419	0.308729	0.224269	0.026017	0.286411	...	0.308475	0.055872	0.197862	0.131367	0.152449	0.084456	0.293293	0.056765	0.10
2	0.118076	1.000000	0.099097	0.107680	0.034279	0.152789	0.086705	0.078864	0.068940	0.092399	...	0.086927	0.259636	0.289092	0.318824	0.149105	0.186347	0.168034	0.106748	0.13
3	0.029097	0.099097	1.000000	0.252131	0.026893	0.062539	0.039767	0.089474	0.078162	0.037670	...	0.040918	0.019031	0.065417	0.055373	0.086503	0.018418	0.096993	0.109631	0.09
4	0.011628	0.107680	0.252131	1.000000	0.000000	0.045543	0.078812	0.095354	0.059498	0.053879	...	0.024226	0.050703	0.056561	0.107294	0.098892	0.000000	0.132900	0.142798	0.09
5	0.264677	0.034279	0.026893	0.000000	1.000000	0.202843	0.299619	0.163724	0.038474	0.153021	...	0.262547	0.048524	0.048312	0.022202	0.091910	0.066000	0.156172	0.115842	0.12
6	0.312419	0.152789	0.062539	0.045543	0.202843	1.000000	0.375963	0.131795	0.110944	0.400758	...	0.287549	0.080312	0.162988	0.182856	0.114262	0.092090	0.261859	0.097606	0.20
7	0.308729	0.086705	0.039767	0.078812	0.299619	0.375963	1.000000	0.211282	0.107795	0.328923	...	0.290002	0.074170	0.094619	0.084235	0.115620	0.100625	0.233843	0.039199	0.22
8	0.224269	0.078864	0.089474	0.095354	0.163724	0.131795	0.211282	1.000000	0.037040	0.183375	...	0.165008	0.066843	0.058766	0.068759	0.087159	0.129381	0.188662	0.121223	0.08
9	0.026017	0.068940	0.078162	0.059498	0.038474	0.110944	0.107795	0.037040	1.000000	0.155435	...	0.011708	0.000000	0.101710	0.034568	0.045002	0.052699	0.107486	0.055766	0.07
10	0.286411	0.092399	0.037670	0.053879	0.153021	0.400758	0.328923	0.183375	0.155435	1.000000	...	0.278558	0.049310	0.153506	0.065471	0.060088	0.033686	0.197107	0.085402	0.11

10 rows × 943 columns

In [19]:
```python
#User Based Collaborative Filter using Weighted Mean Ratings
def cf_user_wmean(user_id, movie_id):

    #Check if movie_id exists in r_matrix
    if movie_id in r_matrix:

        #Get the similarity scores for the user in question with every other user
        sim_scores = cosine_sim[user_id]

        #Get the user ratings for the movie in question
        m_ratings = r_matrix[movie_id]

        #Extract the indices containing NaN in the m_ratings series
        idx = m_ratings[m_ratings.isnull()].index

        #Drop the NaN values from the m_ratings Series
        m_ratings = m_ratings.dropna()

        #Drop the corresponding cosine scores from the sim_scores series
        sim_scores = sim_scores.drop(idx)

        #Compute the final weighted mean
        wmean_rating = np.dot(sim_scores, m_ratings)/ sim_scores.sum()

    else:
        #Default to a rating of 3.0 in the absence of any information
        wmean_rating = 3.0

    return wmean_rating
```

In [20]:
```python
score(cf_user_wmean)
```

Out[20]:
```
1.0174483808407588
```

Demographics

In [21]:
```python
#Merge the original users dataframe with the training set
merged_df = pd.merge(X_train, users)

merged_df.head()
```

Out[21]:

	user_id	movie_id	rating	age	sex	occupation	zip_code
0	889	684	2	24	M	technician	78704
1	889	279	2	24	M	technician	78704
2	889	29	3	24	M	technician	78704
3	889	190	3	24	M	technician	78704
4	889	232	3	24	M	technician	78704

In [24]:
```python
#Compute the mean rating of every movie by gender
gender_mean = merged_df[['movie_id', 'sex', 'rating']].groupby(['movie_id', 'sex'])['rating'].mean()
```

In [25]:
```python
#Set the index of the users dataframe to the user_id
users = users.set_index('user_id')
```

In [26]:
```python
def cf_gender(user_id, movie_id):

    #Check if movie_id exists in r_matrix (or training set)
    if movie_id in r_matrix:
        #Identify the gender of the user
        gender = users.loc[user_id]['sex']

        #Check if the gender has rated the movie
        if gender in gender_mean[movie_id]:

            #Compute the mean rating given by that gender to the movie
            gender_rating = gender_mean[movie_id][gender]

        else:
            gender_rating = 3.0

    else:
        #Default to a rating of 3.0 in the absence of any information
        gender_rating = 3.0

    return gender_rating
```

In [27]:
```python
score(cf_gender)
```

Out[27]:
```
1.0330308800874282
```

In [28]:
```python
#Compute the mean rating by gender and occupation
gen_occ_mean = merged_df[['sex', 'rating', 'movie_id', 'occupation']].pivot_table(
    values='rating', index='movie_id', columns=['occupation', 'sex'], aggfunc='mean')

gen_occ_mean.head()
```

Out[28]:

occupation	administrator		artist		doctor	educator		engineer		entertainment	...	salesman		scientist		student		technician		writer	
sex	F	M	F	M	M	F	M	F	M	F	...	F	M	F	M	F	M	F	M	F	M
movie_id																					
1	4.0	4.222222	4.25	3.500000	3.666667	3.50	3.923077	4.0	3.970588	5.0	...	4.0	4.000000	3.5	3.888889	3.833333	3.709091	4.0	4.200000	4.166667	3.142857
2	3.0	3.750000	NaN	NaN	NaN	NaN	3.250000	NaN	3.363636	NaN	...	NaN	NaN	NaN	NaN	2.333333	3.333333	NaN	2.714286	5.000000	2.666667
3	3.5	2.500000	NaN	NaN	NaN	4.00	2.500000	NaN	3.625000	NaN	...	NaN	1.000000	NaN	NaN	2.000000	3.217391	NaN	4.000000	NaN	1.000000
4	3.0	3.888889	NaN	4.666667	3.000000	2.75	3.636364	NaN	3.555556	NaN	...	4.0	3.666667	NaN	3.600000	3.285714	3.724138	NaN	3.200000	4.250000	3.500000
5	4.0	2.333333	NaN	NaN	NaN	4.00	1.500000	NaN	2.666667	NaN	...	NaN	NaN	NaN	3.500000	4.333333	3.272727	NaN	3.333333	4.000000	2.666667

5 rows × 41 columns

In [29]:
```python
#Gender and Occupation Based Collaborative Filter using Mean Ratings
def cf_gen_occ(user_id, movie_id):

    #Check if movie_id exists in gen_occ_mean
    if movie_id in gen_occ_mean.index:

        #Identify the user
        user = users.loc[user_id]

        #Identify the gender and occupation
        gender = user['sex']
        occ = user['occupation']

        #Check if the occupation has rated the movie
        if occ in gen_occ_mean.loc[movie_id]:

            #Check if the gender has rated the movie
            if gender in gen_occ_mean.loc[movie_id][occ]:

                #Extract the required rating
                rating = gen_occ_mean.loc[movie_id][occ][gender]

                #Default to 3.0 if the rating is null
                if np.isnan(rating):
                    rating = 3.0

                return rating

    #Return the default rating
    return 3.0
```

In [30]:
```python
score(cf_gen_occ)
```

Out[30]:
```
1.1391976012043645
```

In [31]:
```python
pip install scikit-surprise
```

```
Requirement already satisfied: scikit-surprise in c:\users\juand\appdata\local\programs\python\python38\lib\site-packages (1.1.1)Note: you may need to restart
the kernel to use updated packages.
Requirement already satisfied: joblib>=0.11 in c:\users\juand\appdata\local\programs\python\python38\lib\site-packages (from scikit-surprise) (0.17.0)
Requirement already satisfied: numpy>=1.11.2 in c:\users\juand\appdata\local\programs\python\python38\lib\site-packages (from scikit-surprise) (1.19.2)
Requirement already satisfied: scipy>=1.0.0 in c:\users\juand\appdata\local\programs\python\python38\lib\site-packages (from scikit-surprise) (1.5.3)
Requirement already satisfied: six>=1.10.0 in c:\users\juand\appdata\roaming\python\python38\site-packages (from scikit-surprise) (1.15.0)
```

In [35]:
```python
#Import the required classes and methods from the surprise library
from surprise import Reader, Dataset, KNNBasic
from surprise.model_selection import cross_validate

#Define a Reader object
#The Reader object helps in parsing the file or dataframe containing ratings
reader = Reader()

#Create the dataset to be used for building the filter
data = Dataset.load_from_df(ratings, reader)

#Define the algorithm object; in this case kNN
knn = KNNBasic()

#Evaluate the performance in terms of RMSE
cross_validate(knn, data, measures=['RMSE'], cv=5, verbose=True)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9703  0.9818  0.9866  0.9760  0.9770  0.9783  0.0055
Fit time          1.41    1.54    1.38    1.20    1.04    1.32    0.18
Test time         6.98    6.11    4.85    5.19    5.30    5.69    0.77
```

Out[35]:
```
{'test_rmse': array([0.9703318 , 0.98181256, 0.98657855, 0.97602956, 0.97696081]),
 'fit_time': (1.408003330230713,
  1.5440914630889893,
  1.383986473083496,
  1.1999926567077637,
  1.0399940013885498),
 'test_time': (6.975981712341309,
  6.108841180801392,
  4.8480000495910645,
  5.191679954528809,
  5.3039960861206055)}
```

In [ ]: