Building user-based recommendation model for Amazon

February 12, 2022

1 Name: Sunil Pradhan

1.1 Project: 3

1.2 Project Name: Building user-based recommendation model for Amazon

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
#reading the dataset

df=pd.read_csv("Amazon - Movies and TV Ratings.csv")
```

[3]:
```
df.head()
```

[3]:
```
          user_id  Movie1  Movie2  Movie3  Movie4  Movie5  Movie6  Movie7  \
0  A3R5OBKS7OM2IR     5.0     5.0     NaN     NaN     NaN     NaN     NaN
1   AH3QC2PC1VTGP     NaN     NaN     2.0     NaN     NaN     NaN     NaN
2  A3LKP6WPMP9UKX     NaN     NaN     NaN     5.0     NaN     NaN     NaN
3   AVIY68KEPQ5ZD     NaN     NaN     NaN     5.0     NaN     NaN     NaN
4  A1CV1WROP5KTTW     NaN     NaN     NaN     NaN     5.0     NaN     NaN

   Movie8  Movie9  …  Movie197  Movie198  Movie199  Movie200  Movie201  \
0     NaN     NaN  …       NaN       NaN       NaN       NaN       NaN
1     NaN     NaN  …       NaN       NaN       NaN       NaN       NaN
2     NaN     NaN  …       NaN       NaN       NaN       NaN       NaN
3     NaN     NaN  …       NaN       NaN       NaN       NaN       NaN
4     NaN     NaN  …       NaN       NaN       NaN       NaN       NaN

   Movie202  Movie203  Movie204  Movie205  Movie206
0       NaN       NaN       NaN       NaN       NaN
1       NaN       NaN       NaN       NaN       NaN
2       NaN       NaN       NaN       NaN       NaN
3       NaN       NaN       NaN       NaN       NaN
```

```
4       NaN       NaN       NaN       NaN       NaN

[5 rows x 207 columns]
```

[4]:
```
#shape of the dataset

df.shape
```

[4]:
```
(4848, 207)
```

[5]:
```
#checking the info of dataset

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4848 entries, 0 to 4847
Columns: 207 entries, user_id to Movie206
dtypes: float64(206), object(1)
memory usage: 7.7+ MB
```

[6]:
```
#checking the statistical description

df.describe()
```

[6]:
```
       Movie1  Movie2  Movie3  Movie4     Movie5  Movie6  Movie7  Movie8  \
count     1.0     1.0     1.0     2.0  29.000000     1.0     1.0     1.0
mean      5.0     5.0     2.0     5.0   4.103448     4.0     5.0     5.0
std       NaN     NaN     NaN     0.0   1.496301     NaN     NaN     NaN
min       5.0     5.0     2.0     5.0   1.000000     4.0     5.0     5.0
25%       5.0     5.0     2.0     5.0   4.000000     4.0     5.0     5.0
50%       5.0     5.0     2.0     5.0   5.000000     4.0     5.0     5.0
75%       5.0     5.0     2.0     5.0   5.000000     4.0     5.0     5.0
max       5.0     5.0     2.0     5.0   5.000000     4.0     5.0     5.0

       Movie9  Movie10  …  Movie197  Movie198  Movie199  Movie200  Movie201  \
count     1.0      1.0  …  5.000000       2.0       1.0  8.000000  3.000000
mean      5.0      5.0  …  3.800000       5.0       5.0  4.625000  4.333333
std       NaN      NaN  …  1.643168       0.0       NaN  0.517549  1.154701
min       5.0      5.0  …  1.000000       5.0       5.0  4.000000  3.000000
25%       5.0      5.0  …  4.000000       5.0       5.0  4.000000  4.000000
50%       5.0      5.0  …  4.000000       5.0       5.0  5.000000  5.000000
75%       5.0      5.0  …  5.000000       5.0       5.0  5.000000  5.000000
max       5.0      5.0  …  5.000000       5.0       5.0  5.000000  5.000000

       Movie202  Movie203  Movie204   Movie205   Movie206
count  6.000000       1.0  8.000000  35.000000  13.000000
mean   4.333333       3.0  4.375000   4.628571   4.923077
```

```
std     1.632993       NaN  1.407886   0.910259   0.277350
min     1.000000       3.0  1.000000   1.000000   4.000000
25%     5.000000       3.0  4.750000   5.000000   5.000000
50%     5.000000       3.0  5.000000   5.000000   5.000000
75%     5.000000       3.0  5.000000   5.000000   5.000000
max     5.000000       3.0  5.000000   5.000000   5.000000

[8 rows x 206 columns]
```

[7]:
```
#making transpose

describe=df.describe().T
describe
```

[7]:
```
           count      mean       std  min   25%  50%  75%  max
Movie1       1.0  5.000000       NaN  5.0  5.00  5.0  5.0  5.0
Movie2       1.0  5.000000       NaN  5.0  5.00  5.0  5.0  5.0
Movie3       1.0  2.000000       NaN  2.0  2.00  2.0  2.0  2.0
Movie4       2.0  5.000000  0.000000  5.0  5.00  5.0  5.0  5.0
Movie5      29.0  4.103448  1.496301  1.0  4.00  5.0  5.0  5.0
…            …        …        …   …    …   …   …   …
Movie202     6.0  4.333333  1.632993  1.0  5.00  5.0  5.0  5.0
Movie203     1.0  3.000000       NaN  3.0  3.00  3.0  3.0  3.0
Movie204     8.0  4.375000  1.407886  1.0  4.75  5.0  5.0  5.0
Movie205    35.0  4.628571  0.910259  1.0  5.00  5.0  5.0  5.0
Movie206    13.0  4.923077  0.277350  4.0  5.00  5.0  5.0  5.0

[206 rows x 8 columns]
```

1.2.1 Finding movies has maximum views

[8]:
```
desc=describe['count'].sort_values(ascending=False).reset_index().
↪rename(columns={'index':'movies'})
```

[9]:
```
desc
```

[9]:
```
       movies   count
0    Movie127  2313.0
1    Movie140   578.0
2     Movie16   320.0
3    Movie103   272.0
4     Movie29   243.0
..        …       …
201   Movie54     1.0
202  Movie116     1.0
203  Movie115     1.0
204   Movie55     1.0
```

```
205     Movie1      1.0

[206 rows x 2 columns]
```

[10]:
```
print(desc['movies'][0], "has the highest views of:", desc['count'][0])
```

```
Movie127 has the highest views of: 2313.0
```

1.2.2 Finding average rating for each movie

[11]:
```
desc1=describe[['mean','count']].reset_index().rename(columns={'index':
↪'movies','mean':'average_rating'}).
↪sort_values(by='average_rating',ascending=False)
```

[12]:
```
#Average rating of each movies

desc1.iloc[:,:-1]
```

[12]:
```
       movies  average_rating
0      Movie1             5.0
65    Movie66             5.0
75    Movie76             5.0
74    Movie75             5.0
73    Movie74             5.0
..        …               …
57    Movie58             1.0
59    Movie60             1.0
153  Movie154             1.0
44    Movie45             1.0
143  Movie144             1.0

[206 rows x 2 columns]
```

[13]:
```
#finding top 5 movies with the maximum rating

desc1.head()
```

[13]:
```
     movies  average_rating  count
0    Movie1             5.0    1.0
65  Movie66             5.0    1.0
75  Movie76             5.0    2.0
74  Movie75             5.0    1.0
73  Movie74             5.0    1.0
```

Seeing the above 5 movies and their average_rating & count, we can't say these are the top 5 movies with highest rating because count value is 1 and 2 only.

For this let us check the statistical description of count column to get mean count rating and filtering according to that

[14]:
```
desc1['count'].describe()
```

[14]:
```
count     206.000000
mean       24.271845
std       168.937841
min         1.000000
25%         1.000000
50%         2.000000
75%         5.000000
max      2313.000000
Name: count, dtype: float64
```

Here we can see that mean=24.271845, Quantile2=2.0 and thier difference is too high.

Let us take the threshold value = 10

[15]:
```
#filtering according to threshold value=10

desc2=desc1[desc1['count']>=10]
desc2.head()
```

[15]:

	movies	average_rating	count
205	Movie206	4.923077	13.0
161	Movie162	4.866667	15.0
139	Movie140	4.833910	578.0
183	Movie184	4.823529	17.0
157	Movie158	4.818182	66.0

Above 5 are the top 5 movies with the maximum rating

1.2.3 Finding top 5 movies with the least audience

The movie which has rating count value 0 is considered as the lease movie with audience.

let us calculate first the total number of movies having count = 0

[16]:
```
(desc['count']==1).sum()
```

[16]: 89

There are 89 movies which having rating count value = 1

So all those 89 movies will be consider as movies with the least audience

[17]:
```
#movies with least audience

least_aud=desc[desc['count']==1]['movies'].values
least_aud
```

[17]:
```
array(['Movie33', 'Movie165', 'Movie199', 'Movie7', 'Movie21', 'Movie34',
       'Movie8', 'Movie6', 'Movie36', 'Movie37', 'Movie156', 'Movie203',
       'Movie3', 'Movie35', 'Movie10', 'Movie9', 'Movie195', 'Movie20',
       'Movie180', 'Movie153', 'Movie178', 'Movie177', 'Movie176',
       'Movie175', 'Movie187', 'Movie18', 'Movie25', 'Movie17', 'Movie15',
       'Movie171', 'Movie14', 'Movie27', 'Movie13', 'Movie183',
       'Movie154', 'Movie72', 'Movie152', 'Movie58', 'Movie60', 'Movie61',
       'Movie106', 'Movie2', 'Movie63', 'Movie100', 'Movie98', 'Movie64',
       'Movie65', 'Movie66', 'Movie67', 'Movie68', 'Movie69', 'Movie88',
       'Movie87', 'Movie84', 'Movie83', 'Movie71', 'Movie80', 'Movie78',
       'Movie77', 'Movie75', 'Movie74', 'Movie59', 'Movie57', 'Movie149',
       'Movie56', 'Movie147', 'Movie146', 'Movie145', 'Movie144',
       'Movie143', 'Movie142', 'Movie38', 'Movie73', 'Movie41',
       'Movie135', 'Movie42', 'Movie133', 'Movie45', 'Movie46', 'Movie47',
       'Movie48', 'Movie123', 'Movie49', 'Movie50', 'Movie54', 'Movie116',
       'Movie115', 'Movie55', 'Movie1'], dtype=object)
```

[18]:
```
#importing surprise library for recommending

from surprise import Reader, Dataset, SVD, accuracy
from surprise.model_selection import train_test_split, cross_validate
```

[19]:
```
#checking all the column names

df.columns
```

[19]:
```
Index(['user_id', 'Movie1', 'Movie2', 'Movie3', 'Movie4', 'Movie5', 'Movie6',
       'Movie7', 'Movie8', 'Movie9',
       …
       'Movie197', 'Movie198', 'Movie199', 'Movie200', 'Movie201', 'Movie202',
       'Movie203', 'Movie204', 'Movie205', 'Movie206'],
      dtype='object', length=207)
```

[20]:
```
#using melt() to create a dataframe which will allign the user id, movies and␣
 ↪rating as in columns

df_melt=df.melt(id_vars=df.columns[0], value_vars=df.columns[1:],␣
 ↪var_name='movies', value_name='rating')
df_melt
```

[20]:

	user_id	movies	rating
0	A3R5OBKS7OM2IR	Movie1	5.0
1	AH3QC2PC1VTGP	Movie1	NaN
2	A3LKP6WPMP9UKX	Movie1	NaN
3	AVIY68KEPQ5ZD	Movie1	NaN
4	A1CV1WROP5KTTW	Movie1	NaN
…	…	…	…

```
998683  A1IMQ9WMFYKWH5  Movie206     5.0
998684  A1KLIKPUF5E88I  Movie206     5.0
998685   A5HG6WFZL010D  Movie206     5.0
998686  A3UU690TWXCG1X  Movie206     5.0
998687   AI4J762YI6S06  Movie206     5.0

[998688 rows x 3 columns]
```

[21]:

```
df_melt.head()
```

[21]:

```
          user_id  movies  rating
0  A3R5OBKS7OM2IR  Movie1     5.0
1   AH3QC2PC1VTGP  Movie1     NaN
2  A3LKP6WPMP9UKX  Movie1     NaN
3   AVIY68KEPQ5ZD  Movie1     NaN
4  A1CV1WROP5KTTW  Movie1     NaN
```

[22]:

```
#specifying the rating scale

reader=Reader(rating_scale=(-1,10))

data=Dataset.load_from_df(df_melt, reader=reader)
data
```

[22]:

```
<surprise.dataset.DatasetAutoFolds at 0x1c8444e7c40>
```

[23]:

```
#splitting the dataset into traina nd test dataset

train,test= train_test_split(data, test_size=0.25, random_state=78)
```

[24]:

```
#using SVD for fitting and predicting the data

svd=SVD()
svd.fit(train)
```

[24]:

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x1c84411bf10>
```

[25]:

```
pred=svd.test(test)
```

[26]:

```
#finding rmse and mae value

print(accuracy.rmse(pred))
print("")
print(accuracy.mae(pred))
```

```
RMSE: nan
nan
```

```
MAE:  nan
nan
```

[27]:
```
#making a function for filling nan value with 0, mean & median and using cross␣
↪validation to find the value of RMSA & MAE

def acc(svd, data_frame, min, max):
    reader=Reader(rating_scale=(min,max))
    data=Dataset.load_from_df(data_frame, reader=reader)
    print(cross_validate(svd, data, measures=['RMSE', 'MAE'], cv=3,␣
↪verbose=True))
    print("\n\n")

    user_id=df_melt['user_id'][0]
    movie_id=df_melt['movies'][0]
    rating=df_melt['rating'][0]

    print(svd.predict(user_id, movie_id, rating, verbose=True))

    print("\n\n")
```

[28]:
```
#filling the nan value with 0

acc(SVD(), df_melt.fillna(0), -1, 10)
```

```
Evaluating RMSE, MAE of algorithm SVD on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.2831  0.2778  0.2879  0.2829  0.0041
MAE (testset)     0.0431  0.0416  0.0433  0.0427  0.0008
Fit time          30.08   30.55   30.50   30.38   0.21
Test time         2.51    2.82    2.51    2.61    0.15
{'test_rmse': array([0.28313669, 0.27777637, 0.2879076 ]), 'test_mae':
array([0.04307648, 0.04159981, 0.04329575]), 'fit_time': (30.08289670944214,
30.54972243309021, 30.498133659362793), 'test_time': (2.5106120109558105,
2.823787212371826, 2.50720477104187)}



user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = -0.01
{'was_impossible': False}
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = -0.01
{'was_impossible': False}
```

[29]:
```
#filling the nan value with mean

acc(SVD(), df_melt.fillna(df_melt['rating'].mean()), -1, 10)
```

```
Evaluating RMSE, MAE of algorithm SVD on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.0830  0.0868  0.0882  0.0860  0.0022
MAE (testset)     0.0097  0.0100  0.0100  0.0099  0.0001
Fit time          30.27   32.44   31.06   31.26   0.90
Test time         2.81    2.47    2.83    2.70    0.16
{'test_rmse': array([0.08301379, 0.08675295, 0.08822973]), 'test_mae':
array([0.00974175, 0.01003169, 0.00999947]), 'fit_time': (30.26589322090149,
32.436994552612305, 31.062217473983765), 'test_time': (2.8081953525543213,
2.4700441360473633, 2.8250701427459717)}
```

```
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 4.39
{'was_impossible': False}
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 4.39
{'was_impossible': False}
```

[30]:
```
#filling the nan value with median

acc(SVD(), df_melt.fillna(df_melt['rating'].median()), -1, 10)
```

```
Evaluating RMSE, MAE of algorithm SVD on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.0916  0.0933  0.0931  0.0927  0.0008
MAE (testset)     0.0090  0.0084  0.0085  0.0086  0.0003
Fit time          30.42   31.12   31.37   30.97   0.40
Test time         2.53    2.83    2.48    2.61    0.15
{'test_rmse': array([0.09155033, 0.09334302, 0.0930592 ]), 'test_mae':
array([0.00901893, 0.00835193, 0.0085142 ]), 'fit_time': (30.42193293571472,
31.123861074447632, 31.368030786514282), 'test_time': (2.5319323539733887,
2.829014778137207, 2.4834303855895996)}
```

```
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 5.00
{'was_impossible': False}
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 5.00
{'was_impossible': False}
```

Here, we found that filling nan with mean value gives us the lowest rmse value.

So we prefer mean as filling the nan value

[36]:
```
from surprise.model_selection import GridSearchCV
```

[35]:
```
#using GRidSearchCV to find best parameters and accuracy

param_grid={'n_epochs':[20,30],
           'lr_all':[0.005,0.01],
           'n_factors':[50,100]}
param_grid
```

[35]:
```
{'n_epochs': [20, 30], 'lr_all': [0.005, 0.01], 'n_factors': [50, 100]}
```

[38]:
```
gs=GridSearchCV(SVD, param_grid, measures=['rmse', 'mae'], cv=3)
data1=Dataset.load_from_df(df_melt.fillna(df_melt['rating'].mean()),␣
 ↪reader=reader)
gs.fit(data1)
```

[39]:
```
#checking best RMSE & MAE value

gs.best_score
```

[39]:
```
{'rmse': 0.08503491801805137, 'mae': 0.008390146354550663}
```

[40]:
```
#checking best param grid

gs.best_params
```

[40]:
```
{'rmse': {'n_epochs': 30, 'lr_all': 0.01, 'n_factors': 50},
 'mae': {'n_epochs': 30, 'lr_all': 0.01, 'n_factors': 50}}
```

[ ]:
```

```