ML_Amazon_recommenderSystem

December 11, 2020

[6]:
```
#MachineLearning Final Project Submission.
#Building user-based recommendation model for Amazon

import pandas as pd
import numpy as np
```

[7]:
```
#Read the "Amazon - Movies and TV Ratings.csv" file from the folder into the␣
 ↪program.
DF_Amazon_moviesData = pd.read_csv("Amazon - Movies and TV Ratings.csv")
```

[8]:
```
#Analysis of the Data

DF_Amazon_moviesData.head()
```

[8]:

	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]

[9]:
```
DF_Amazon_moviesData.tail()
```

[9]:
```
             user_id  Movie1  Movie2  Movie3  Movie4  Movie5  Movie6  Movie7  \
4843  A1IMQ9WMFYKWH5     NaN     NaN     NaN     NaN     NaN     NaN     NaN
4844  A1KLIKPUF5E88I     NaN     NaN     NaN     NaN     NaN     NaN     NaN
4845   A5HG6WFZLO10D     NaN     NaN     NaN     NaN     NaN     NaN     NaN
4846  A3UU690TWXCG1X     NaN     NaN     NaN     NaN     NaN     NaN     NaN
4847   AI4J762YI6S06     NaN     NaN     NaN     NaN     NaN     NaN     NaN

      Movie8  Movie9  …  Movie197  Movie198  Movie199  Movie200  Movie201  \
4843     NaN     NaN  …       NaN       NaN       NaN       NaN       NaN
4844     NaN     NaN  …       NaN       NaN       NaN       NaN       NaN
4845     NaN     NaN  …       NaN       NaN       NaN       NaN       NaN
4846     NaN     NaN  …       NaN       NaN       NaN       NaN       NaN
4847     NaN     NaN  …       NaN       NaN       NaN       NaN       NaN

      Movie202  Movie203  Movie204  Movie205  Movie206
4843       NaN       NaN       NaN       NaN       5.0
4844       NaN       NaN       NaN       NaN       5.0
4845       NaN       NaN       NaN       NaN       5.0
4846       NaN       NaN       NaN       NaN       5.0
4847       NaN       NaN       NaN       NaN       5.0

[5 rows x 207 columns]
```

[10]:
```
DF_Amazon_moviesData.columns
```

[10]:
```
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)
```

[11]:
```
DF_Amazon_moviesData.describe()
```

[11]:
```
        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]
```

[12]:
```
DF_Amazon_moviesData.shape
```

[12]:
```
(4848, 207)
```

[13]:
```
DF_Amazon_moviesData.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
```

[14]:
```
#Check for missing values
print('Number of missing values across columns: \n',DF_Amazon_moviesData.
 ↪isnull().sum())
```

```
Number of missing values across columns:
 user_id        0
Movie1      4847
Movie2      4847
Movie3      4847
Movie4      4846
            …
Movie202    4842
Movie203    4847
Movie204    4840
Movie205    4813
```

```
Movie206    4835
Length: 207, dtype: int64
```

[15]:
```
#creating Copy for backup.
DF_Amazon_moviesData_mainCopy = DF_Amazon_moviesData.copy()
```

[16]:
```
#Which movies have maximum views/ratings?

DF_Amazon_moviesData.describe().T["count"].sort_values(ascending=False)[:10].
↪to_frame()
```

[16]:

	count
Movie127	2313.0
Movie140	578.0
Movie16	320.0
Movie103	272.0
Movie29	243.0
Movie91	128.0
Movie92	101.0
Movie89	83.0
Movie158	66.0
Movie108	54.0

[17]:
```
#Define the top 5 movies with the maximum ratings.

DF_Amazon_moviesData.drop('user_id',axis=1).sum().sort_values(ascending=False)[:
↪5].to_frame()
```

[17]:

	0
Movie127	9511.0
Movie140	2794.0
Movie16	1446.0
Movie103	1241.0
Movie29	1168.0

[18]:
```
#What is the average rating for each movie?

DF_Amazon_moviesData.drop('user_id',axis=1).mean().
↪sort_values(ascending=False)[:10].to_frame()
```

[18]:

	0
Movie1	5.0
Movie55	5.0
Movie131	5.0
Movie132	5.0
Movie133	5.0
Movie63	5.0

```
Movie135  5.0
Movie136  5.0
Movie61   5.0
Movie57   5.0
```

[19]:
```
#Define the top 5 movies with the least audience.

DF_Amazon_moviesData.describe().T["count"].sort_values(ascending=True)[:5].
↪to_frame()
```

[19]:
```
          count
Movie1      1.0
Movie71     1.0
Movie145    1.0
Movie69     1.0
Movie68     1.0
```

[20]:
```
#Recommandation Model

from surprise import Reader
from surprise import accuracy
from surprise import Dataset
from surprise.model_selection import train_test_split

DF_Amazon_moviesData.columns
```

[20]:
```
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)
```

[21]:
```
#re-formatting Data for processing using melt command
melt_DF_Amazon_moviesData = DF_Amazon_moviesData.melt(id_vars =␣
↪DF_Amazon_moviesData.columns[0], value_vars = DF_Amazon_moviesData.columns[1:
↪],var_name="Movie",value_name="Rating")
melt_DF_Amazon_moviesData
```

[21]:
```
                user_id     Movie  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   A5HG6WFZLO10D  Movie206     5.0
998686  A3UU690TWXCG1X  Movie206     5.0
998687   AI4J762YI6S06  Movie206     5.0

[998688 rows x 3 columns]
```

[22]:
```
reader_data = Reader(rating_scale=(-1,10))

Model_data = Dataset.load_from_df(melt_DF_Amazon_moviesData.
 ↪fillna(0),reader=reader_data)

trainset_data,testset_data = train_test_split(Model_data,test_size=0.25)
```

[23]:
```
from surprise import SVD

svd_model = SVD()

svd_model.fit(trainset_data)
```

[23]:
```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7f45769efa50>
```

[24]:
```
predictions_value = svd_model.test(testset_data)
```

[25]:
```
accuracy.rmse(predictions_value)
```

```
RMSE: 0.2764
```

[25]:
```
0.27641456133608944
```

[26]:
```
from surprise.model_selection import cross_validate

cross_validate(svd_model,Model_data,measures=['RMSE','MAE'], cv=6, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 6 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Fold 6  Mean    Std
RMSE (testset)    0.2807  0.2806  0.2752  0.2847  0.2672  0.2751  0.2772  0.0056
MAE (testset)     0.0408  0.0409  0.0401  0.0414  0.0386  0.0400  0.0403  0.0009
Fit time          44.60   45.00   45.09   45.03   45.09   45.16   44.99   0.18
Test time         1.47    1.44    1.77    1.83    1.46    1.44    1.57    0.17
```

[26]:
```
{'test_rmse': array([0.28074814, 0.28062692, 0.27515281, 0.28469775, 0.26715025,
        0.27507702]),
 'test_mae': array([0.04076578, 0.04091246, 0.04005124, 0.04137867, 0.03863338,
        0.03999113]),
 'fit_time': (44.600287437438965,
```

```
 44.99930191040039,
 45.08531212806702,
 45.03063941001892,
 45.09192180633545,
 45.15942406654358),
'test_time': (1.4742791652679443,
 1.4358928203582764,
 1.7678749561309814,
 1.8346896171569824,
 1.455456256866455,
 1.4367806911468506)}
```

[27]:
```
def repeat_function(algo_type,dataframe_,min_,max_):
    reader_data = Reader(rating_scale=(min_,max_))
    Model_data = Dataset.load_from_df(dataframe_,reader=reader_data)
    model = algo_type
    print(cross_validate(model,Model_data,measures=['RMSE','MAE'], cv=6,␣
 ↪verbose=True))
    print("#"*10)

    #using first data for testing

    user_id = 'A3R5OBKS7OM2IR'
    muvi_id = 'Movie1'
    r_ui = 5.0
    print(model.predict(user_id,muvi_id,r_ui=r_ui,verbose=True))
    print("#"*10)
    print()
```

[28]:
```
DF_Amazon_moviesData = DF_Amazon_moviesData.iloc[:1212,:50]

melt_DF_Amazon_moviesData = DF_Amazon_moviesData.melt(id_vars =␣
 ↪DF_Amazon_moviesData.columns[0], value_vars = DF_Amazon_moviesData.columns[1:
 ↪],var_name="Movie",value_name="Rating")
```

repeat_function(SVD(),melt_DF_Amazon_moviesData.fillna(0),-1,10)

repeat_function(SVD(),melt_DF_Amazon_moviesData.fillna(melt_DF_Amazon_moviesData.mean()),-1,10)

repeat_function(SVD(),melt_DF_Amazon_moviesData.fillna(melt_DF_Amazon_moviesData.median()),-1,10)

[29]:
```
repeat_function(SVD(),melt_DF_Amazon_moviesData.fillna(0),-1,10)

repeat_function(SVD(),melt_DF_Amazon_moviesData.
 ↪fillna(melt_DF_Amazon_moviesData.mean()),-1,10)
```

```
repeat_function(SVD(),melt_DF_Amazon_moviesData.
 ↪fillna(melt_DF_Amazon_moviesData.median()),-1,10)
```

```
Evaluating RMSE, MAE of algorithm SVD on 6 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Fold 6  Mean    Std
RMSE (testset)    0.4546  0.4730  0.4331  0.4350  0.4553  0.4160  0.4445  0.0186
MAE (testset)     0.1014  0.1040  0.0969  0.0951  0.0999  0.0904  0.0980  0.0044
Fit time          2.60    2.59    2.59    2.59    2.59    2.58    2.59    0.01
Test time         0.06    0.34    0.06    0.06    0.06    0.06    0.11    0.10
{'test_rmse': array([0.45464997, 0.47304059, 0.4330634 , 0.43495016, 0.45530742,
       0.41595412]), 'test_mae': array([0.10139883, 0.10395743, 0.09693233,
0.09514346, 0.09985255,
       0.09044445]), 'fit_time': (2.595305919647217, 2.5905494689941406,
2.593817949295044, 2.5914175510406494, 2.588618278503418, 2.575411796569824),
'test_time': (0.06348419189453125, 0.3417680263519287, 0.06343603134155273,
0.06342005729675293, 0.06397223472595215, 0.06341433525085449)}
##########
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 0.40
{'was_impossible': False}
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 0.40
{'was_impossible': False}
##########

Evaluating RMSE, MAE of algorithm SVD on 6 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Fold 6  Mean    Std
RMSE (testset)    0.0853  0.0742  0.0907  0.0986  0.0949  0.0909  0.0891  0.0078
MAE (testset)     0.0176  0.0172  0.0171  0.0183  0.0178  0.0181  0.0177  0.0004
Fit time          2.60    2.62    2.60    2.60    2.61    2.59    2.60    0.01
Test time         0.06    0.06    0.06    0.06    0.06    0.06    0.06    0.00
{'test_rmse': array([0.08530153, 0.07421082, 0.09068428, 0.0985756 , 0.09487633,
       0.09093157]), 'test_mae': array([0.01759638, 0.01716047, 0.01708935,
0.01828463, 0.01776256,
       0.01811932]), 'fit_time': (2.598290205001831, 2.618053913116455,
2.603726387023926, 2.5977673530578613, 2.609282970428467, 2.5895614624023438),
'test_time': (0.0636749267578125, 0.06352877616882324, 0.06345820426940918,
0.06343245506286621, 0.06428742408752441, 0.0637211799621582)}
##########
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 4.62
{'was_impossible': False}
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 4.62
{'was_impossible': False}
##########

Evaluating RMSE, MAE of algorithm SVD on 6 split(s).
```

```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Fold 6  Mean    Std
RMSE (testset)    0.0856  0.0868  0.1109  0.1008  0.1122  0.0830  0.0966  0.0120
MAE (testset)     0.0164  0.0165  0.0174  0.0169  0.0175  0.0162  0.0168  0.0005
Fit time          2.56    2.59    2.60    2.58    2.58    2.57    2.58    0.01
Test time         0.35    0.06    0.06    0.06    0.06    0.06    0.11    0.11
{'test_rmse': array([0.08555155, 0.08679429, 0.11094865, 0.10084422, 0.11223112,
       0.08303408]), 'test_mae': array([0.01640263, 0.01652215, 0.01743749,
0.0169265 , 0.0174692 ,
       0.01619697]), 'fit_time': (2.558335781097412, 2.585143566131592,
2.599632740020752, 2.5807809829711914, 2.581191062927246, 2.5729546546936035),
'test_time': (0.34549403190612793, 0.06328988075256348, 0.06400299072265625,
0.0637211799621582, 0.06375575065612793, 0.06354069709777832)}
##########
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 5.01
{'was_impossible': False}
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 5.01
{'was_impossible': False}
##########
```

[30]:
```
from surprise.model_selection import GridSearchCV

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

gridsearch_CV = GridSearchCV(SVD,parameter_grid,measures=['rmse','mae'],cv=3)

gridsearch_CV.fit(Model_data)
```

[31]:
```
gridsearch_CV.best_score
```

[31]:
```
{'rmse': 0.2779665929552559, 'mae': 0.04025276192296653}
```

[32]:
```
reader_data = Reader(rating_scale=(-1,10))

Model_data = Dataset.load_from_df(melt_DF_Amazon_moviesData.
 ↪fillna(melt_DF_Amazon_moviesData.median()),reader=reader_data)

trainset_data,testset_data = train_test_split(Model_data,test_size=0.25)

from surprise import SVD

svd_model = SVD()

svd_model.fit(trainset_data)
```

```
predictions_value = svd_model.test(testset_data)

accuracy.rmse(predictions_value)
```

```
RMSE: 0.1019
```

[32]: 0.10194496164128161