DSC630_WK10.02_Kim-Schreck

May 16, 2024

1 DSC630_WK10.02_Kim-Schreck

[1]:
```
# imports

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from datetime import datetime
from surprise import Reader, Dataset, SVD
from surprise.model_selection import cross_validate
from sklearn.metrics import r2_score
import warnings
warnings.filterwarnings("ignore")
```

[2]:
```
# 10.02.01-01
# read csv
# assign variable for dataset
# importing ratings.csv as dt01
# dt01

dt01 = pd.read_csv('ratings.csv')
```

[3]:
```
# 10.02.01-02
# return first ten rows
# dt01

dt01.head(10)
```

[3]:

	userId	movieId	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
```

[4]:
```
# 10.02.01-03
# return last ten rows
# dt01

dt01.tail(10)
```

[4]:

	userId	movieId	rating	timestamp
100826	610	162350	3.5	1493849971
100827	610	163937	3.5	1493848789
100828	610	163981	3.5	1493850155
100829	610	164179	5.0	1493845631
100830	610	166528	4.0	1493879365
100831	610	166534	4.0	1493848402
100832	610	168248	5.0	1493850091
100833	610	168250	5.0	1494273047
100834	610	168252	5.0	1493846352
100835	610	170875	3.0	1493846415

[5]:
```
# 10.02.01-04
# return dimensions
# despite being the smallest dataset, there are 100836 rows
# dt01

print(dt01.shape)
```

```
(100836, 4)
```

[6]:
```
# 10.02.01-05
# rename columns for better clarity of content
# I have a naming system prioritizing categories from large to small
# dt01

dt01.rename(columns = {'userId':'id_user'}, inplace = True)
dt01.rename(columns = {'movieId':'id_movie'}, inplace = True)
```

[7]:
```
# 10.02.01-06
# return columns
# confirming renamed columns
# dt01

dt01.columns
```

[7]:
```
Index(['id_user', 'id_movie', 'rating', 'timestamp'], dtype='object')
```

[8]:
```
# 10.02.02-01
# read csv
# assign variable for dataset
# importing movies.csv as dt02
# dt02

dt02 = pd.read_csv('movies.csv')
```

[9]:
```
# 10.02.02-02
# return first ten rows
# dt02

dt02.head(10)
```

[9]:

	movieId	…	genres
0	1	…	Adventure|Animation|Children|Comedy|Fantasy
1	2	…	Adventure|Children|Fantasy
2	3	…	Comedy|Romance
3	4	…	Comedy|Drama|Romance
4	5	…	Comedy
5	6	…	Action|Crime|Thriller
6	7	…	Comedy|Romance
7	8	…	Adventure|Children
8	9	…	Action
9	10	…	Action|Adventure|Thriller

[10 rows x 3 columns]

[10]:
```
# 10.02.02-03
# return last ten rows
# dt02

dt02.tail(10)
```

[10]:

	movieId	…	genres
9732	193565	…	Action|Animation|Comedy|Sci-Fi
9733	193567	…	Animation|Drama
9734	193571	…	Comedy|Drama
9735	193573	…	Animation
9736	193579	…	Documentary
9737	193581	…	Action|Animation|Comedy|Fantasy
9738	193583	…	Animation|Comedy|Fantasy
9739	193585	…	Drama
9740	193587	…	Action|Animation
9741	193609	…	Comedy

[10 rows x 3 columns]

[11]:
```
# 10.02.02-04
# return dimensions
# despite being the smallest dataset, there are 9742 rows
# dt02

print(dt02.shape)
```

```
(9742, 3)
```

[12]:
```
# 10.02.02-05
# rename columns for better clarity of content
# I have a naming system prioritizing categories from large to small
# dt02

dt02.rename(columns = {'movieId':'id_movie'}, inplace = True)
```

[13]:
```
# 10.02.02-06
# return columns
# confirming renamed columns
# dt02

dt02.columns
```

[13]:
```
Index(['id_movie', 'title', 'genres'], dtype='object')
```

[14]:
```
# 10.02.03-01
# merge csvs: ratings and movies (dt01 and dt02)
# combine datasets for model preparation
# assigned variable dt03 to merged datasets
# dt03

dt03 = pd.merge(dt01, dt02, on = 'id_movie')
```

[15]:
```
# 10.02.03-02
# return first ten rows
# dt03

dt03.head(10)
```

[15]:

	id_user	…	genres
0	1	…	Adventure|Animation|Children|Comedy|Fantasy
1	5	…	Adventure|Animation|Children|Comedy|Fantasy
2	7	…	Adventure|Animation|Children|Comedy|Fantasy
3	15	…	Adventure|Animation|Children|Comedy|Fantasy
4	17	…	Adventure|Animation|Children|Comedy|Fantasy
5	18	…	Adventure|Animation|Children|Comedy|Fantasy
6	19	…	Adventure|Animation|Children|Comedy|Fantasy
7	21	…	Adventure|Animation|Children|Comedy|Fantasy

```
8       27  …  Adventure|Animation|Children|Comedy|Fantasy
9       31  …  Adventure|Animation|Children|Comedy|Fantasy

[10 rows x 6 columns]
```

[16]:
```
# 10.02.03-03
# return last ten rows
# dt03

dt03.tail(10)
```

[16]:
```
        id_user  …                  genres
100826      610  …        Action|Adventure
100827      610  …             Documentary
100828      610  …     Comedy|Drama|Horror
100829      610  …                  Horror
100830      610  …  Action|Comedy|Thriller
100831      610  …         Action|Thriller
100832      610  …      Action|Crime|Drama
100833      610  …   Action|Drama|Thriller
100834      610  …         Horror|Thriller
100835      610  …                  Horror

[10 rows x 6 columns]
```

[17]:
```
# 10.02.03-04
# return dimensions of merged data
# merged data contains 100836 rows
# dt03

print(dt03.shape)
```

```
(100836, 6)
```

[18]:
```
# 10.02.03-05
# return summary of merged data
# there appears to be no missing values
# dt03

print(dt03.info)
```

```
<bound method DataFrame.info of         id_user  …
genres
0             1  …  Adventure|Animation|Children|Comedy|Fantasy
1             5  …  Adventure|Animation|Children|Comedy|Fantasy
2             7  …  Adventure|Animation|Children|Comedy|Fantasy
3            15  …  Adventure|Animation|Children|Comedy|Fantasy
4            17  …  Adventure|Animation|Children|Comedy|Fantasy
```

```
…          …  …                                        …
100831       610  …                              Action|Thriller
100832       610  …                         Action|Crime|Drama
100833       610  …                      Action|Drama|Thriller
100834       610  …                            Horror|Thriller
100835       610  …                                      Horror

[100836 rows x 6 columns]>
```

[19]:
```
# 10.02.03-05
# rename columns for better clarity of content
# I have a naming system prioritizing categories from large to small
# dt03

dt03.rename(columns = {'userId':'id_user'}, inplace = True)
dt03.rename(columns = {'movieId':'id_movie'}, inplace = True)
```

[20]:
```
# 10.02.03-06
# return columns
# confirming renamed columns
# dt03

dt03.columns
```

[20]:
```
Index(['id_user', 'id_movie', 'rating', 'timestamp', 'title', 'genres'],
dtype='object')
```

[21]:
```
# 10.02.04-01
# split train / test, 75%
# thought it would be reasonable to split 25/75
# dt03

dt03_trn, dt03_tst = train_test_split(dt03, test_size = 0.25, random_state = 36)
```

[22]:
```
# 10.02.05-01
# assign variable for reader
# using reader library to create a rating scale of 0-5
# dt03

dt03_rdr = Reader(rating_scale = (0, 5))
```

[23]:
```
# 10.02.06-01
# load training data to ds
# load three columns (id_user, id_movie, and rating) into a dataset for model␣
 ↪as dt03_trn_ds
# dt03
```

```
dt03_trn_ds = Dataset.load_from_df(dt03_trn[['id_user', 'id_movie', 'rating']],␣
 ↪dt03_rdr)
```

[24]:
```
# 10.02.07-01
# build SVD algorithm for filtering as dt_03_mdl
# singular value decomposition algorithm to break down amount of data
# singular value decomposition algorithm to make predictions on unrated data
# dt03

dt_03_mdl = SVD()
```

[25]:
```
# 10.02.07-02
# return training data
# dt03

print(dt03_trn_ds)
```

```
<surprise.dataset.DatasetAutoFolds object at 0x127246f50>
```

[26]:
```
# 10.02.08-01
# evaluate model using cross-validation
# calculate rmse and mae for accuracy
# dt03

cross_validate(dt_03_mdl, dt03_trn_ds, measures = ['RMSE', 'MAE'], cv = 5,␣
 ↪verbose = True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8869  0.8836  0.8892  0.8869  0.8899  0.8873  0.0022
MAE (testset)     0.6828  0.6782  0.6847  0.6850  0.6862  0.6834  0.0028
Fit time          0.82    0.82    0.79    0.80    1.01    0.85    0.08
Test time         0.09    0.08    0.08    0.10    0.17    0.10    0.03
```

[26]:
```
{'test_rmse': array([0.88691472, 0.88355275, 0.88918442, 0.88687308,
0.88993359]),
 'test_mae': array([0.68284253, 0.67823272, 0.68474359, 0.68502713,
0.68617215]),
 'fit_time': (0.8241171836853027,
  0.816547155380249,
  0.7902400493621826,
  0.7953040599822998,
  1.0119659900665283),
 'test_time': (0.09066390991210938,
  0.07746601104736328,
  0.07874894142150879,
  0.10126209259033203,
```

```
0.16796302795410156)}
```

[27]:
```python
# 10.02.09-01
# create function rec_mv to return movie titles and ratings
# return movies from title input
# return ratings of input
# predict movie rating of user
# sort ratings predictions descending
# return top recommended movies
# dt02, dt03

def rec_mv(dt_03_ttl_mv, dt_03_mdl, dt02, dt03, n = 12):
    dt_03_id_mv = dt02[dt02['title'] == dt_03_ttl_mv]['id_movie'].iloc[0]
    dt_03_rt_mv = dt03[dt03['id_movie'] == dt_03_id_mv]
    dt_03_id_usr = 0
    dt_03_lst_pdct = []
    for dt_03_id_mv in dt03['id_movie'].unique():
        dt_03_pdct = dt_03_mdl.predict(dt_03_id_usr, dt_03_id_mv)
        dt_03_lst_pdct.append((dt_03_id_mv, dt_03_pdct.est))
    dt_03_lst_pdct.sort(key = lambda x: x[1], reverse = True)
    dt_03_lst_rec_mv = []
    for dt_03_id_mv, _ in dt_03_lst_pdct[:n]:
        dt_03_rec_mv = dt02[dt02['id_movie'] == dt_03_id_mv]['title'].iloc[0]
        dt_03_lst_rec_mv.append(dt_03_rec_mv)
    return dt_03_lst_rec_mv
```

[28]:
```python
# 10.02.10-01
# test rec_mv function using random title
# dt03

dt_03_sel_mv = 'Fight Club (1999)'
dt_03_lst_rec_mv = rec_mv(dt_03_sel_mv, dt_03_mdl, dt02, dt03)
```

[29]:
```python
# 10.02.10-02
# return test result
# returns ten movie titles that are similar to the input title
# dt03

print(f"Ten selected movies based on '{dt_03_sel_mv}':")
for mv in dt_03_lst_rec_mv:
    print(mv)
```

```
Ten selected movies based on 'Fight Club (1999)':
Shawshank Redemption, The (1994)
Lawrence of Arabia (1962)
Amelie (Fabuleux destin d'Amélie Poulain, Le) (2001)
Usual Suspects, The (1995)
Godfather, The (1972)
```

```
Bridge on the River Kwai, The (1957)
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)
Fight Club (1999)
Boondock Saints, The (2000)
Reservoir Dogs (1992)
Dark Knight, The (2008)
Schindler's List (1993)
```

[30]:
```python
# 10.02.11-01
# create function to exit_program
# quits program
# dt03

def exit_program():
    print("Exiting the program...")
    sys.exit()
```

[ ]:
```python
# 10.02.11-02
# create function to to return input title as get_inp_ttl
# return result from user input
# dt03

def get_inp_ttl():
    while True:
        try:
            inp_itm = input('Please enter a movie title:')
            if inp_itm == 'Fight Club (1999)':
                print(f"Ten selected movies based on '{dt_03_sel_mv}':",␣
 ↪dt_03_lst_rec_mv)
            elif inp_itm != 'Fight Club (1999)':
                print('Entry is invalid.')
            else:
                print('Entry is invalid.')
        except:
            print('The program has quit.')
            exit_program()
get_inp_ttl()
```

```
Please enter a movie title: Fight Club (1999)

Ten selected movies based on 'Fight Club (1999)': ['Shawshank Redemption, The
(1994)', 'Lawrence of Arabia (1962)', "Amelie (Fabuleux destin d'Amélie Poulain,
Le) (2001)", 'Usual Suspects, The (1995)', 'Godfather, The (1972)', 'Bridge on
the River Kwai, The (1957)', 'Dr. Strangelove or: How I Learned to Stop Worrying
and Love the Bomb (1964)', 'Fight Club (1999)', 'Boondock Saints, The (2000)',
'Reservoir Dogs (1992)', 'Dark Knight, The (2008)', "Schindler's List (1993)"]
```

2 DSC630_WK10.03_Kim-Schreck

[ ]:
```
# 10.03.01-01

# DSC630_WK10.03_Kim-Schreck_peer_review_of_Arhia_Dominquez.docx
```