HomeWork

December 9, 2018

```
In [1]: from sqlalchemy import create_engine
        from sqlalchemy.exc import ResourceClosedError
        from sqlalchemy.types import VARCHAR
        from functools import partial
        import pandas as pd
        from scipy.spatial import distance
        from sklearn.neighbors import NearestNeighbors
        from IPython.display import display, HTML, Markdown, Latex
        from sklearn.utils.extmath import randomized_svd
        import numpy as np
```

0.1 Reading the data from database into a pandas dataframe

```
In [2]: #connection
        def DatabaseConnect(username, password, schema):
            conn_str = "mysql+pymysql://{username}:{password}@localhost/{schema}?charset=utf8&u
                            .format(username=username, password=password,schema=schema
            engine = create_engine(conn_str, pool_recycle=1800)
            return engine
        RecSysConnect = partial(DatabaseConnect, 'root', 'mysql-password', 'recsys')
        e = RecSysConnect()
```

```
In [3]: #read ratings
        sql_cmt = "select userId, movieId, rating from ml100k_ratings;"
        raw_rating = pd.read_sql(sql_cmt, con=e)
        rating = raw_rating.pivot(index="userId", columns="movieId", values="rating")
        rating.head(10)
```

```
Out[3]: movieId  1     2     3     4     5     6     7     8     9     10    ...   \
        userId                                                               ...
        1        5.0   3.0   4.0   3.0   3.0   5.0   4.0   1.0   5.0   3.0   ...
        2        4.0   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   2.0   ...
        3        NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   ...
        4        NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   ...
        5        4.0   3.0   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   ...
        6        4.0   NaN   NaN   NaN   NaN   NaN   2.0   4.0   4.0   NaN   ...
        7        NaN   NaN   NaN   5.0   NaN   NaN   5.0   5.0   5.0   4.0   ...
        8        NaN   NaN   NaN   NaN   NaN   NaN   3.0   NaN   NaN   NaN   ...
```

```
9          NaN   NaN   NaN   NaN   NaN   5.0   4.0   NaN   NaN   NaN  ...
10         4.0   NaN   NaN   4.0   NaN   NaN   4.0   NaN   4.0   NaN  ...

movieId  1673  1674  1675  1676  1677  1678  1679  1680  1681  1682
userId
1          NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
2          NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
3          NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
4          NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
5          NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
6          NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
7          NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
8          NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
9          NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN
10         NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN   NaN

[10 rows x 1682 columns]
```

In [4]:
```
#read holdout data
sql_cmt = "select userId from ml100k_ratings group by userId order by count(*) DESC li
user_100 = pd.read_sql(sql_cmt, con=e)['userId']
sql_cmt = "select movieId from ml100k_ratings group by movieId order by count(*) DESC l
movie_10 = pd.read_sql(sql_cmt, con=e)['movieId']
holdout = rating.loc[user_100,movie_10]
```

In [5]:
```
rating.loc[user_100,movie_10]=np.nan
```

0.2 Calculation of similarities

In [6]:
```
import sklearn.metrics.pairwise

def center(df):
    return df.sub( df.mean(axis=1), axis=0 )

def cosine(df, axis=0):
    dff = df.fillna(0)
    if axis == 0: # Columns
        return pd.DataFrame(sklearn.metrics.pairwise.cosine_similarity(dff.T), index=d
    else:
        return pd.DataFrame(sklearn.metrics.pairwise.cosine_similarity(dff),   index=d
```

In [7]:
```
r_cent = center(rating)
userSim = rating.T.corr(method='pearson')
#userSim = cosine(r_cent, axis=1)
itemSim = cosine(r_cent)
display(userSim.head(),itemSim.head())
```

```
userId       1          2          3          4          5          6          7    \
userId
```

```
1       1.000000  0.156868 -0.069171 -0.666667  0.386678  0.298236  0.260460
2       0.156868  1.000000  0.067420  0.148522  0.327327  0.446269  0.381875
3      -0.069171  0.067420  1.000000 -0.262600       NaN -0.109109  0.104828
4      -0.666667  0.148522 -0.262600  1.000000  1.000000 -0.581318 -0.660529
5       0.386678  0.327327       NaN  1.000000  1.000000  0.241817  0.170291

userId       8         9        10      ...        934       935       936  \
userId                                  ...
1       0.694108 -0.301511 -0.124713    ...   0.066802 -0.252820  0.435300
2       0.585491  0.242536  0.668145    ...   0.021007 -0.271163  0.214017
3       0.291937       NaN  0.311086    ...        NaN       NaN -0.045162
4       0.642938       NaN -0.301511    ...   0.500000       NaN -0.203653
5       0.537400  0.577350  0.087343    ...   0.229532 -0.500000  0.439286

userId       937       938       939       940       941       942       943
userId
1      -0.044483  0.157353  0.584437  0.257908  0.000000 -0.202777  0.048947
2       0.561645  0.331587  0.000000 -0.011682 -0.062017  0.085960  0.479702
3       0.000000 -0.137523       NaN -0.104678  1.000000 -0.011792       NaN
4            NaN  0.375000       NaN  0.850992  1.000000  0.412568       NaN
5       0.608581  0.484211  0.880705  0.027038  0.468521  0.318163  0.346234

[5 rows x 943 columns]


movieId      1         2         3         4         5         6         7     \
movieId
1        1.000000 -0.023667 -0.042012  0.012120  0.007037 -0.002027  0.020884
2       -0.023667  1.000000  0.031568  0.030879  0.013452 -0.010600 -0.053574
3       -0.042012  0.031568  1.000000 -0.118348  0.012564  0.054819 -0.087453
4        0.012120  0.030879 -0.118348  1.000000 -0.149492 -0.022921  0.005652
5        0.007037  0.013452  0.012564 -0.149492  1.000000 -0.041295 -0.043651

movieId      8         9        10      ...        1673  1674      1675  \
movieId                                 ...
1        0.100244 -0.046859  0.002856    ...     0.066344   0.0  0.000000
2       -0.007358 -0.103660 -0.020773    ...     0.000000   0.0  0.000000
3       -0.151500 -0.066376 -0.049901    ...     0.000000   0.0  0.000000
4        0.111663  0.029485  0.009932    ...     0.000000   0.0 -0.114413
5        0.012695 -0.061202 -0.033263    ...     0.000000   0.0  0.000000

movieId      1676      1677  1678  1679  1680      1681      1682
movieId
1        0.000000  0.012261   0.0   0.0   0.0  0.000000  0.000000
2        0.000000  0.000000   0.0   0.0   0.0  0.003661  0.034941
3        0.000000  0.201111   0.0   0.0   0.0  0.000000  0.031866
4       -0.114413  0.090004   0.0   0.0   0.0  0.002727 -0.048234
5        0.000000  0.000000   0.0   0.0   0.0  0.000000  0.043673
```

```
[5 rows x 1682 columns]
```

0.3 Recommenders

0.3.1 - randomized_svd

```
In [8]: def svd_pred(rating,holdout):
            rating.columns = rating.columns.map(str)
            means = rating.mean(axis=1)
            cent  = rating.sub( means, axis=0 )
            U_, Sigma, VT_ = randomized_svd(cent.to_sparse().to_coo().tocsc(), n_components=2,
            U = pd.DataFrame(U_, index=rating.index)
            VT = pd.DataFrame(VT_, columns=rating.columns)
            full = U.mul(Sigma).dot(VT)
            full.columns=full.columns.map(int)
            means = rating.mean(axis=1).loc[user_100]
            prediction = full.loc[user_100,movie_10].add(means,axis=0)
            y_true=[]
            y_predicted=[]
            for u in holdout.index:
                for m in holdout.columns:
                    if (holdout.isna().loc[u,m]==False):
                        y_true=y_true+[holdout.loc[u,m]]
                        y_predicted=y_predicted+[prediction.loc[u,m]]
            return y_true, y_predicted
```

```
In [9]: y_true_svd, y_pred_svd = svd_pred(rating,holdout)
```

```
In [10]: pd.DataFrame([y_true_svd,y_pred_svd])
```

```
Out[10]:          0         1         2         3        4         5         6    \
         0  5.000000  5.000000  5.000000  4.000000  2.00000  3.000000  3.000000
         1  3.969673  3.624942  2.092731  3.329952  2.95191  3.541682  2.897582

                 7         8         9      ...          745       746      747  \
         0  3.000000  3.000000  3.000000    ...     5.000000  4.000000  5.00000
         1  2.428885  2.977192  2.705344    ...     3.488522  3.394138  3.49022

                 748      749       750       751      752       753       754
         0  5.000000  5.00000  4.000000  4.000000  4.00000  4.000000  1.000000
         1  3.973007  3.68861  4.086564  3.667382  3.31407  3.672904  3.285864

         [2 rows x 755 columns]
```

0.3.2 - user-based

```
In [11]: def recommendUser(df, user, item, k=3, weighted=True):
             knn = userSim.loc[user].drop(user).nlargest(k)
```

```
        norm = knn.sum()
        rating = 0
        for u in knn.index:
            rating += df.loc[u, item] * knn[u]
        rating = np.clip( rating / norm, 1, 5 )
        return (rating)
```

```
In [12]: def user_based(rating,holdout):
             rating.columns = rating.columns.map(int)
             y_true=[]
             y_predicted=[]
             for u in holdout.index:
                 for i in holdout.columns:
                     if (holdout.isna().loc[u,i]==False):
                         y_true=y_true+[holdout.loc[u,i]]
                         y_predicted=y_predicted+[recommendUser(rating, u, i, 2)]
             return y_true,y_predicted
         y_true_user, y_pred_user = user_based(rating,holdout)
         df=pd.DataFrame([y_true_user,y_pred_user]).dropna(axis=1)
         y_true_user = df.iloc[0]
         y_pred_user = df.iloc[1]
         display(df)
```

```
   2         7    9    11        17        18        21   27   28        46  \
0  5.0  3.000000  3.0  3.0  2.000000  3.000000  1.000000  4.0  4.0  4.000000
1  3.0  3.989795  4.0  4.0  2.946724  3.579914  3.473362  3.0  4.0  4.463657

  ...   683  688  699  707  717  718  720  722  734  752
0 ...   5.0  3.0  5.0  5.0  3.0  5.0  3.0  4.0  3.0  4.0
1 ...   3.5  2.0  3.0  3.5  3.0  3.0  2.5  4.5  3.5  2.5

[2 rows x 104 columns]
```

0.3.3 - item-based

```
In [13]: def recommendItem(df,user, item, k=2, weighted=True):
             knn = itemSim.loc[item][ df.loc[user].notnull() ]
             knn = knn[ knn> 0 ].drop(item, errors='ignore').nlargest(k)
             if weighted:
                norm = knn.sum()
             else:
                knn.values.fill(1/len(knn))
                norm = 1
             ratings = 0
             for i in knn.index:
                 ratings += df.loc[user,i] * knn[i]
             ratings = np.clip( ratings / norm, 1, 5 )
```

```
        return (ratings)
    def item_based(rating,holdout):
        rating.columns = rating.columns.map(int)
        y_true=[]
        y_predicted=[]
        for u in holdout.index:
            for i in holdout.columns:
                if (holdout.isna().loc[u,i]==False):
                    y_true=y_true+[holdout.loc[u,i]]
                    y_predicted=y_predicted+[recommendItem(rating, u, i)]
        return y_true,y_predicted
    y_true_item, y_pred_item = item_based(rating,holdout)
    pd.DataFrame([y_true_item,y_pred_item])
```

Out[13]:

```
     0    1         2         3    4         5         6         7         8  \
0  5.0  5.0  5.000000  4.000000  2.0  3.000000  3.000000  3.000000  3.000000
1  5.0  5.0  2.857781  3.531249  4.0  4.048189  3.601238  2.499696  2.429392

          9    ...     745  746       747       748       749       750  \
0  3.000000    ...     5.0  4.0  5.000000  5.000000  5.000000  4.000000
1  2.587803    ...     4.0  4.0  4.417514  3.531249  3.490975  4.551081

        751       752       753       754
0  4.000000  4.000000  4.000000  1.000000
1  3.601238  3.473194  3.468862  2.835027

[2 rows x 755 columns]
```

0.4 Evaluation: MSE MAE

In [14]:

```
from sklearn.metrics import mean_squared_error,mean_absolute_error
def evaluation(y_true, y_predicted):
    result=pd.DataFrame([y_true,y_predicted])
    print(result)
    #MAE=np.mean(np.abs(result[0] - result[1]))/755
    MAE = mean_absolute_error(y_true,y_predicted)
    RMSE = np.sqrt(mean_squared_error(y_true,y_predicted))
    #RMSE=np.mean((result[0] - result[1])**2)/755
    display("MAE",MAE,"RMSE",RMSE)
```

In [15]:

```
evaluation(y_true_svd, y_pred_svd)
```

```
         0         1         2         3        4         5         6  \
0  5.000000  5.000000  5.000000  4.000000  2.00000  3.000000  3.000000
1  3.969673  3.624942  2.092731  3.329952  2.95191  3.541682  2.897582

         7         8         9    ...          745       746      747  \
0  3.000000  3.000000  3.000000    ...     5.000000  4.000000  5.00000
1  2.428885  2.977192  2.705344    ...     3.488522  3.394138  3.49022
```

```
        748      749       750       751      752       753       754
0  5.000000  5.00000  4.000000  4.000000  4.00000  4.000000  1.000000
1  3.973007  3.68861  4.086564  3.667382  3.31407  3.672904  3.285864

[2 rows x 755 columns]
```

```
'MAE'
```

```
0.7386856980861768
```

```
'RMSE'
```

```
0.9117497430960818
```

In [16]: `evaluation(y_true_item, y_pred_item)`

```
   0    1         2         3    4         5         6         7         8    \
0  5.0  5.0  5.000000  4.000000  2.0  3.000000  3.000000  3.000000  3.000000
1  5.0  5.0  2.857781  3.531249  4.0  4.048189  3.601238  2.499696  2.429392

        9      ...     745  746       747       748       749       750  \
0  3.000000    ...     5.0  4.0  5.000000  5.000000  5.000000  4.000000
1  2.587803    ...     4.0  4.0  4.417514  3.531249  3.490975  4.551081

        751       752       753       754
0  4.000000  4.000000  4.000000  1.000000
1  3.601238  3.473194  3.468862  2.835027

[2 rows x 755 columns]
```

```
'MAE'
```

```
0.6783970962926593
```

```
'RMSE'
```

```
0.9412219376564158
```

In [17]: `evaluation(y_true_user, y_pred_user)`

```
   2         7    9    11         17         18         21   27   28         46  \
0  5.0  3.000000  3.0  3.0  2.000000  3.000000  1.000000  4.0  4.0  4.000000
1  3.0  3.989795  4.0  4.0  2.946724  3.579914  3.473362  3.0  4.0  4.463657

  ...   683  688  699  707  717  718  720  722  734  752
0 ...   5.0  3.0  5.0  5.0  3.0  5.0  3.0  4.0  3.0  4.0
1 ...   3.5  2.0  3.0  3.5  3.0  3.0  2.5  4.5  3.5  2.5

[2 rows x 104 columns]
```

```
'MAE'
```

```
1.0512574811196467
```

```
'RMSE'
```

```
1.2742844501470791
```

In [14]:
```python
#Use suprise package
from surprise import SVD
from surprise.model_selection import cross_validate
from surprise import Reader,Dataset
reader = Reader(rating_scale=(1, 5))
# The columns must correspond to user id, item id and ratings (in that order).
data = Dataset.load_from_df(raw_rating, reader)
cross_validate(SVD(), data, measures=['RMSE', 'MAE'], cv=2, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 2 split(s).

                  Fold 1  Fold 2  Mean    Std
RMSE (testset)    0.9598  0.9560  0.9579  0.0019
MAE (testset)     0.7583  0.7563  0.7573  0.0010
Fit time          3.38    4.16    3.77    0.39
Test time         0.66    0.59    0.63    0.04
```

Out[14]:
```
{'fit_time': (3.375916004180908, 4.163803815841675),
 'test_mae': array([0.75829125, 0.7563168 ]),
 'test_rmse': array([0.95982148, 0.95600433]),
 'test_time': (0.6646459102630615, 0.5870378017425537)}
```