In [1]:
```python
# Importing standard libraries
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```python
# Importing ratings file
ratings = pd.read_csv('ratings.csv')
ratings.head()
```

Out[2]:

| | user_id | movie_id | rating | unix_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 [3]:
```python
# Create a number of ratings column to see how many ratings each movie
 has
ratings_over_fifty = pd.DataFrame(ratings.groupby('movie_id')['rating']
.count())
# Rename the column to number_of_ratings
ratings_over_fifty.columns = ['number_of_ratings']
# Get only the movies that have more than 50 ratings
ratings_over_fifty = ratings_over_fifty[ratings_over_fifty['number_of_r
atings']>50]
# Reset index so that movie_id is not index, put an artificial index va
lue
ratings_over_fifty.reset_index(inplace=True)
```

```python
# See the movies with over 50 ratings
ratings_over_fifty.head(20)
```

Out[3]:

| | movie_id | number_of_ratings |
|---|---|---|
| **0** | 1 | 452 |
| **1** | 2 | 131 |
| **2** | 3 | 90 |
| **3** | 4 | 209 |
| **4** | 5 | 86 |
| **5** | 7 | 392 |
| **6** | 8 | 219 |
| **7** | 9 | 299 |
| **8** | 10 | 89 |
| **9** | 11 | 236 |
| **10** | 12 | 267 |
| **11** | 13 | 184 |
| **12** | 14 | 183 |
| **13** | 15 | 293 |
| **14** | 17 | 92 |
| **15** | 19 | 69 |
| **16** | 20 | 72 |
| **17** | 21 | 84 |
| **18** | 22 | 297 |
| **19** | 23 | 182 |

In [4]:
```
# Define function to check if a movie in the main dataset has over 50 r
atings or not
def rows_above_under_fifty(row):
    if np.any(row == ratings_over_fifty['movie_id']):
        return True
    return False
```

In [5]:
```
# Apply that function on the movies column, assigns the True or False v
alues in a new Series object
series_above_under_fifty = ratings['movie_id'].apply(rows_above_under_f
ifty)
# It can be seen that the 1st, 2nd movies have over 50 ratings, whereas
 the 3rd one does not
# 1st corresponds to movie_id 242
# 2nd corresponds to movie_id 302
# 3rd corresponds to movie_id 377
series_above_under_fifty.head()
```

Out[5]:
```
0     True
1     True
2    False
3     True
4     True
Name: movie_id, dtype: bool
```

In [6]:
```
# Let's see if that is the case
# Prints out that it has 117 ratings
print(ratings_over_fifty[ratings_over_fifty['movie_id'] == 242])
# Prints out that is has 297 ratings
print(ratings_over_fifty[ratings_over_fifty['movie_id'] == 302])
# Returns empty dataframe => it is not in the ratings_over_fifty list =
> it does not have over 50 ratings
print(ratings_over_fifty[ratings_over_fifty['movie_id'] == 377])
```

```
     movie_id  number_of_ratings
209       242                117
     movie_id  number_of_ratings
257       302                297
```

```
Empty DataFrame
Columns: [movie_id, number_of_ratings]
Index: []
```

In [7]:
```
# Filter ratings dataset so that only the movies with above 50 rating r
emain
ratings_filtered = ratings[series_above_under_fifty]
ratings_filtered.head()
```

Out[7]:

| | user_id | movie_id | rating | unix_timestamp |
|---|---|---|---|---|
| **0** | 196 | 242 | 3 | 881250949 |
| **1** | 186 | 302 | 3 | 891717742 |
| **3** | 244 | 51 | 2 | 880606923 |
| **4** | 166 | 346 | 1 | 886397596 |
| **5** | 298 | 474 | 4 | 884182806 |

In [8]:
```
# Split the ratings dataframe into train and test set
from sklearn.model_selection import train_test_split
# Model with the whole dataset
ratings_train, ratings_test = train_test_split(ratings, test_size=0.15,
 random_state=1)
# Model with only the movies that have above 50 rating
#ratings_train, ratings_test = train_test_split(ratings_filtered, test_
size=0.15, random_state=1)
```

In [9]:
```
# Turicreate is a high-level machine learning library created by Apple
import turicreate as tc
ratings_train = tc.SFrame(ratings_train)
ratings_test = tc.SFrame(ratings_test)
```

```
/home/vanko/anaconda3/lib/python3.6/site-packages/h5py/__init__.py:36:
FutureWarning: Conversion of the second argument of issubdtype from `fl
oat` to `np.floating` is deprecated. In future, it will be treated as `
```

```
np.float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

In [10]:
```
# A simple popularity model is trained, which recommends movies based o
n their popularity
# item_id parameter specifies the column name that will be recommended,
 namely, movie_id
popularity_model = tc.popularity_recommender.create(ratings_train, user
_id='user_id',
                                                            item_id='mo
vie_id', target='rating')
```

```
Recsys training: model = popularity

Warning: Ignoring columns unix_timestamp;

    To use these columns in scoring predictions, use a model that allow
s the use of additional features.

Preparing data set.

    Data has 85000 observations with 943 users and 1653 items.

    Data prepared in: 0.334914s

85000 observations to process; with 1653 unique items.
```

In [11]:
```
# Recommend the top 5 movies to users 1, 2, 3, 4, 5
popularity_recomm = popularity_model.recommend(users=[1,2,3,4,5],k=5)
popularity_recomm.print_rows(num_rows=25)
```

```
+---------+----------+-------+------+
| user_id | movie_id | score | rank |
+---------+----------+-------+------+
|    1    |   1189   |  5.0  |  1   |
|    1    |   1599   |  5.0  |  2   |
|    1    |   1293   |  5.0  |  3   |
|    1    |   1643   |  5.0  |  4   |
|    1    |   1536   |  5.0  |  5   |
|    2    |   1189   |  5.0  |  1   |
```

```
|     2    |   1599   |  5.0  |  2   |
|     2    |   1293   |  5.0  |  3   |
|     2    |   1643   |  5.0  |  4   |
|     2    |   1536   |  5.0  |  5   |
|     3    |   1189   |  5.0  |  1   |
|     3    |   1599   |  5.0  |  2   |
|     3    |   1293   |  5.0  |  3   |
|     3    |   1643   |  5.0  |  4   |
|     3    |   1536   |  5.0  |  5   |
|     4    |   1189   |  5.0  |  1   |
|     4    |   1599   |  5.0  |  2   |
|     4    |   1293   |  5.0  |  3   |
|     4    |   1643   |  5.0  |  4   |
|     4    |   1536   |  5.0  |  5   |
|     5    |   1189   |  5.0  |  1   |
|     5    |   1599   |  5.0  |  2   |
|     5    |   1293   |  5.0  |  3   |
|     5    |   1643   |  5.0  |  4   |
|     5    |   1536   |  5.0  |  5   |
+----------+----------+-------+------+
[25 rows x 4 columns]
```

In [12]:

```
# The above was a simple popularity model. Now we will build a collabor
ative-filtering model.
# Ranking factorization recommender trains a model to predict a rating
 for
# each possible combination of users and movies. The internal coefficie
nts of the model are
# learned from known ratings of users on movies. Recommendations are th
en based on these ratings.
# Training the model
item_sim_model = tc.ranking_factorization_recommender.create(ratings_tr
ain, user_id='user_id',
                                                             item_id='m
ovie_id', target='rating')
```

```
Recsys training: model = ranking_factorization_recommender
```

```
Preparing data set.

    Data has 85000 observations with 943 users and 1653 items.

    Data prepared in: 0.240837s

Training ranking_factorization_recommender for recommendations.

+--------------------------------+------------------------------------
-------------+----------+

| Parameter                      | Description
             | Value    |

+--------------------------------+------------------------------------
-------------+----------+

| num_factors                    | Factor Dimension
             | 32       |

| regularization                 | L2 Regularization on Factors
             | 1e-09    |

| solver                         | Solver used for training
             | adagrad  |

| linear_regularization          | L2 Regularization on Linear Coeffici
ents         | 1e-09    |

| ranking_regularization         | Rank-based Regularization Weight
             | 0.25     |

| max_iterations                 | Maximum Number of Iterations
             | 25       |

+--------------------------------+------------------------------------
-------------+----------+

  Optimizing model using SGD; tuning step size.
```

```
Using 10625 / 85000 points for tuning the step size.
+---------+------------------+--------------------------------------
--+
| Attempt | Initial Step Size | Estimated Objective Value
  |
+---------+------------------+--------------------------------------
--+
| 0       | 16.6667          | Not Viable
  |
| 1       | 4.16667          | Not Viable
  |
| 2       | 1.04167          | Not Viable
  |
| 3       | 0.260417         | Not Viable
  |
| 4       | 0.0651042        | 1.8657
  |
| 5       | 0.0325521        | 1.8702
  |
| 6       | 0.016276         | 1.98718
  |
| 7       | 0.00813802       | 2.12851
  |
+---------+------------------+--------------------------------------
--+
| Final   | 0.0651042        | 1.8657
```

```
|
+---------+-----------------+--------------------------------------
--+
Starting Optimization.
+---------+-------------+-----------------+----------------------+-
------------+
| Iter.   | Elapsed Time | Approx. Objective | Approx. Training RMSE |
Step Size   |
+---------+-------------+-----------------+----------------------+-
------------+
| Initial | 17.426ms     | 2.48549           | 1.12591               |
            |
+---------+-------------+-----------------+----------------------+-
------------+
| 1       | 828.95ms     | 2.12678           | 1.14161               |
0.0651042   |
| 2       | 1.50s        | 1.90197           | 1.06083               |
0.0651042   |
| 3       | 2.17s        | 1.81637           | 1.0283                |
0.0651042   |
| 4       | 3.06s        | 1.73893           | 1.00472               |
0.0651042   |
| 5       | 4.05s        | 1.6847            | 0.987224              |
0.0651042   |
| 10      | 7.50s        | 1.54326           | 0.940059              |
0.0651042   |
```

```
| 15      | 10.17s      | 1.46666          | 0.909931            |
0.0651042   |

| 20      | 14.98s      | 1.41824          | 0.883675            |
0.0651042   |

| 25      | 19.04s      | 1.43903          | 0.869355            |
0.0651042   |

+---------+-------------+------------------+---------------------+-
------------+

Optimization Complete: Maximum number of passes through the data reache
d.

Computing final objective value and training RMSE.

       Final objective value: 1.45394

       Final training RMSE: 0.852866
```

In [13]:

```
# Making recommendations
item_sim_recomm = item_sim_model.recommend(users=[1,2,3,4,5],k=5)
item_sim_recomm.print_rows(num_rows=25)
```

```
+---------+----------+--------------------+------+
| user_id | movie_id |       score        | rank |
+---------+----------+--------------------+------+
|    1    |   408    | 4.633140248337915  |  1   |
|    1    |   474    | 4.430625391284158  |  2   |
|    1    |   919    | 4.391392669731905  |  3   |
|    1    |   483    | 4.2822603407298825 |  4   |
|    1    |   238    | 4.257882935086419  |  5   |
|    2    |   269    | 4.561351833025148  |  1   |
|    2    |   258    | 4.416115162054231  |  2   |
|    2    |   283    | 4.395786491314103  |  3   |
|    2    |   127    | 4.348903534094026  |  4   |
|    2    |   124    | 4.308760699907472  |  5   |
|    3    |    50    | 4.862409320751359  |  1   |
```

```
|    3    |   127   | 4.486713168302705  |  2   |
|    3    |    98   | 4.334899989048173  |  3   |
|    3    |    56   | 4.300042775551011  |  4   |
|    3    |   257   | 4.239566472450425  |  5   |
|    4    |   302   | 4.755937409678628  |  1   |
|    4    |   100   | 4.604095113555124  |  2   |
|    4    |   313   | 4.6038982394134305 |  3   |
|    4    |   258   | 4.565941047946145  |  4   |
|    4    |   127   | 4.564202514568498  |  5   |
|    5    |    7    | 4.349293303767373  |  1   |
|    5    |   175   |  4.19093973664396  |  2   |
|    5    |   408   | 4.190826487818887  |  3   |
|    5    |   173   | 4.131918025294473  |  4   |
|    5    |    12   | 4.0241669538413785 |  5   |
+---------+---------+--------------------+------+
[25 rows x 4 columns]
```

In [14]:

```
# Evaluate RMSE (Root Mean Square Error), the lower it is, the better.
 A lower score means the actual data points
# are closer to the regression line (therefore the regression line's pr
edicted value is closer to the actual value)
# For visual explanation refer to:
# https://www.khanacademy.org/math/ap-statistics/bivariate-data-ap/asse
ssing-fit-least-squares-regression/v/standard-dev-residuals
item_sim_model.evaluate_rmse(ratings_test, target='rating')
# For this model, on the testing data, the RMSE is ~1.01, which is not
 too bad. It is also quite close to
# the RMSE of the model on the training data, which is ~0.85.
```

Out[14]:

```
{'rmse_by_user': Columns:
        user_id int
        rmse    float
        count   int

 Rows: 936

 Data:
 +---------+--------------------+-------+
```

```
 | user_id |        rmse        | count |
 +---------+--------------------+-------+
 |   747   | 1.2981214691745575 |   45  |
 |   118   | 1.0451524006516273 |   8   |
 |   153   | 1.508314253741147  |   2   |
 |   660   | 1.0182323399319324 |   35  |
 |    92   | 0.9108288857742471 |   47  |
 |   264   | 1.3253550391043698 |   19  |
 |   690   | 0.8469186934862357 |   16  |
 |   839   | 1.3639768389329134 |   7   |
 |   837   | 1.0468211086952406 |   6   |
 |   208   | 1.0593900404122296 |   1   |
 +---------+--------------------+-------+
 [936 rows x 3 columns]
 Note: Only the head of the SFrame is printed.
 You can use print_rows(num_rows=m, num_columns=n) to print more rows a
nd columns.,
 'rmse_by_item': Columns:
        movie_id        int
        rmse    float
        count   int

 Rows: 1366

 Data:
 +----------+---------------------+-------+
 | movie_id |         rmse        | count |
 +----------+---------------------+-------+
 |   973    |  0.8836474840855688 |   1   |
 |   1270   |  0.8200935647819125 |   2   |
 |   1611   |  0.3223828077892839 |   2   |
 |   747    |  0.8211008559973578 |   20  |
 |   118    |  0.958870429970247  |   44  |
 |   153    |  1.051357770442147  |   35  |
 |   660    |  0.863244055470406  |   30  |
 |   1236   | 0.14044750002808737 |   1   |
 |    92    |  1.2084345090230597 |   17  |
 |   1657   |  0.6132284969121984 |   1   |
 +----------+---------------------+-------+
```

```
 [1366 rows x 3 columns]
 Note: Only the head of the SFrame is printed.
 You can use print_rows(num_rows=m, num_columns=n) to print more rows a
nd columns.,
 'rmse_overall': 1.0156485870814331}
```