Hybrid_Recommendation_Engine

July 1, 2019

1 Hybrid recommendation system

Photo by Felix Mooneeram

1.1 Origin of dataset

MovieLens data sets were collected by the GroupLens Research Project at the University of Minnesota.

This data set consists of: * 100,000 ratings (1-5) from 943 users on 1682 movies. * Each user has rated at least 20 movies. * Simple demographic info for the users (age, gender, occupation, zip)

The data was collected through the MovieLens web site (movielens.umn.edu) during the seven-month period from September 19th, 1997 through April 22nd, 1998. This data has been cleaned up - users who had less than 20 ratings or did not have complete demographic information were removed from this data set. Detailed descriptions of the data file can be found at the end of this file.

Neither the University of Minnesota nor any of the researchers involved can guarantee the correctness of the data, its suitability for any particular purpose, or the validity of results based on the use of the data set. The data set may be used for any research purposes under the following conditions:

1.2 Definition of a recommandation system

Credits: Wikipedia

It is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. They are primarily used in commercial applications.

Recommender systems are utilized in a variety of areas, and are most commonly recognized as playlist generators for video and music services like Netflix, YouTube and Spotify, product recommenders for services such as Amazon, or content recommenders for social media platforms such as Facebook and Twitter. These systems can operate using a single input, like music, or multiple inputs within and across platforms like news, books, and search queries. There are also popular recommender systems for specific topics like restaurants and online dating.

1.3 Recommandation engine design

There are different approaches, some of them are:

Collaborative filtering

It is based on the assumption that people who agreed in the past will agree in the future, and that they will like similar kinds of items as they liked in the past. The system generates recommendations using only information about rating profiles for different users or items. By locating peer users/items with a rating history similar to the current user or item, they generate recommendations using this neighborhood [...].

Advantage: it does not rely on machine analyzable content and therefore it is capable of accurately recommending complex items such as movies without requiring an "understanding" of the item itself [...].

Content-based filtering

It is based on a description of the item and a profile of the user's preferences. This methods is best suited to situations where there is known data on an item (name, location, description, etc.), but not on the user. Content-based recommenders treat recommendation as a user-specific classification problem and learn a classifier for the user's likes and dislikes based on product features.

In this system, keywords are used to describe the items and a user profile is built to indicate the type of item this user likes. This algorithms try to recommend items that are similar to those that a user liked in the past, or is examining in the present. It does not rely on a user sign-in mechanism to generate this often temporary profile. In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended.[...]

Hybrid recommender systems

Most recommender systems now use a hybrid approach, combining collaborative filtering, content-based filtering, and other approaches.

---

2 Data exploration & preparation

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
from scipy import sparse

#import numpy as np
#import scipy.sparse as sparse
```

2.1 Detailed description of used data files

In [62]:
```
rating_cols = ['user_id', 'movie_id', 'rating', 'unix_timestamp']
df_ratings = pd.read_csv('../input/u.data', sep='\t', names=rating_cols)
df_ratings.shape
```

Out[62]:
```
(100000, 4)
```

In [63]:
```
df_ratings.head()
```

Out[63]:

	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 [64]:
```
users_cols = ['user_id', 'age', 'sex', 'occupation', 'zip_code']
df_users = pd.read_csv('../input/u.user', sep='|', names=users_cols, parse_dates=True)
df_users.head()
```

Out[64]:
```
   user_id  age sex  occupation zip_code
0        1   24   M  technician    85711
1        2   53   F       other    94043
2        3   23   M      writer    32067
3        4   24   M  technician    43537
4        5   33   F       other    15213
```

In [65]:
```
items_cols = ['movie_id' , 'movie_title' , 'release_date' , 'video_release_date' , 'IM
              'Fantasy|', 'Film-Noir|', 'Horror|', 'Musical|', 'Mystery|', 'Romance|'
              'War|', 'Western|']
df_items = pd.read_csv('../input/u.item', sep='|', encoding='latin-1', names=items_col
df_items.head()
```

Out[65]:
```
                movie_title release_date  video_release_date  \
movie_id
1          Toy Story (1995)  01-Jan-1995                 NaN
2          GoldenEye (1995)  01-Jan-1995                 NaN
3         Four Rooms (1995)  01-Jan-1995                 NaN
4         Get Shorty (1995)  01-Jan-1995                 NaN
5            Copycat (1995)  01-Jan-1995                 NaN

                                                     IMDb_URL  unknown|  \
movie_id
1         http://us.imdb.com/M/title-exact?Toy%20Story%2...         0
2         http://us.imdb.com/M/title-exact?GoldenEye%20(...         0
3         http://us.imdb.com/M/title-exact?Four%20Rooms%...         0
4         http://us.imdb.com/M/title-exact?Get%20Shorty%...         0
5         http://us.imdb.com/M/title-exact?Copycat%20(1995)         0

          Action|  Adventure|  Animation|  Children's|  Comedy|    ...      \
movie_id                                                           ...
1               0           0           1            1        1    ...
2               1           1           0            0        0    ...
3               0           0           0            0        0    ...
4               1           0           0            0        1    ...
5               0           0           0            0        0    ...

          Fantasy|  Film-Noir|  Horror|  Musical|  Mystery|  Romance|  \
movie_id
1                0           0        0         0         0         0
2                0           0        0         0         0         0
3                0           0        0         0         0         0
```

```
4                 0            0        0          0          0          0
5                 0            0        0          0          0          0

          Sci-Fi|  Thriller|  War|  Western|
movie_id
1              0          0     0          0
2              0          1     0          0
3              0          1     0          0
4              0          0     0          0
5              0          1     0          0

[5 rows x 23 columns]
```

In [66]:
```
movie_cols = ['movie_id', 'title', 'release_date', 'video_release_date', 'imdb_url']
df_movies = pd.read_csv('../input/u.item', sep='|', names=movie_cols, usecols=range(5
df_movies.shape
```

Out[66]:
```
(1682, 4)
```

In [67]:
```
df_movies.head()
```

Out[67]:
```
                      title release_date  video_release_date  \
movie_id
1          Toy Story (1995)  01-Jan-1995                 NaN
2          GoldenEye (1995)  01-Jan-1995                 NaN
3         Four Rooms (1995)  01-Jan-1995                 NaN
4         Get Shorty (1995)  01-Jan-1995                 NaN
5            Copycat (1995)  01-Jan-1995                 NaN

                                                    imdb_url
movie_id
1         http://us.imdb.com/M/title-exact?Toy%20Story%2...
2         http://us.imdb.com/M/title-exact?GoldenEye%20(...
3         http://us.imdb.com/M/title-exact?Four%20Rooms%...
4         http://us.imdb.com/M/title-exact?Get%20Shorty%...
5         http://us.imdb.com/M/title-exact?Copycat%20(1995)
```

df_movies provides infos for each movie (ie for each line).

df_ratings provides infos for each rating a user have made.

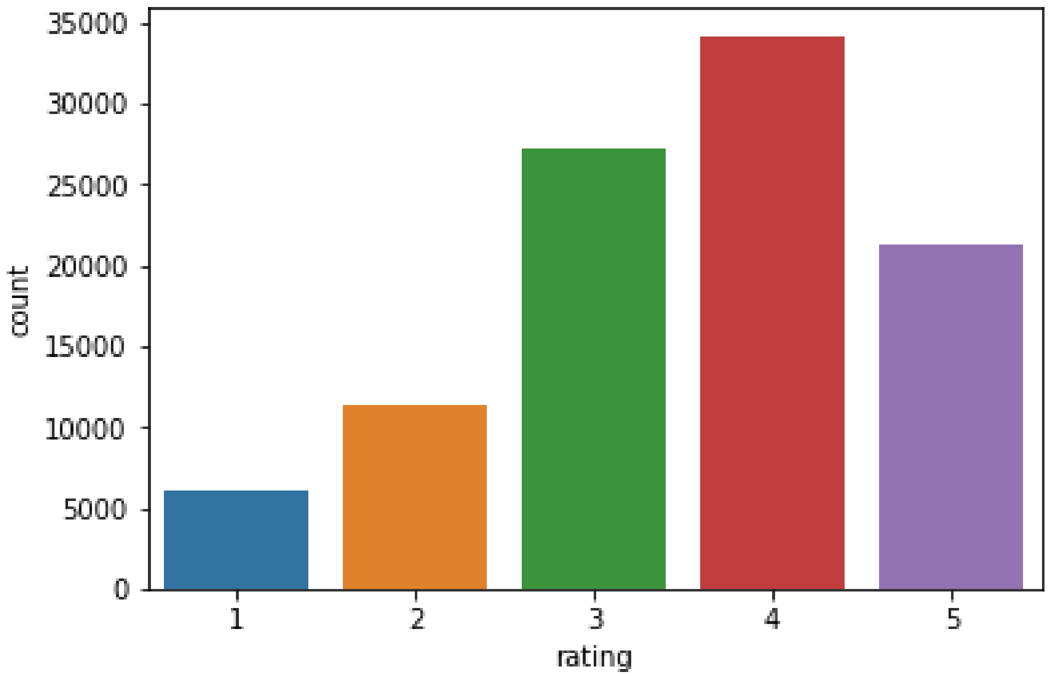
u.data -- The full u data set, 100000 ratings by 943 users on 1682 items. Each user has rated at least 20 movies. Users and items are numbered consecutively from 1. The data is randomly ordered. This is a tab separated list of user id | item id | rating | timestamp. The time stamps are unix seconds since 1/1/1970 UTC

u.item -- Information about the items (movies); this is a tab separated list of movie id | movie title | release date | video release date | IMDb URL | unknown | Action | Adventure | Animation | Children's | Comedy | Crime | Documentary | Drama | Fantasy | Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi | Thriller | War | Western | The last 19 fields are the genres, a 1 indicates the movie is of that genre, a 0 indicates it is not; movies can be in several genres at once. The movie ids are the ones used in the u.data data set.

2.2 Data Visualizations

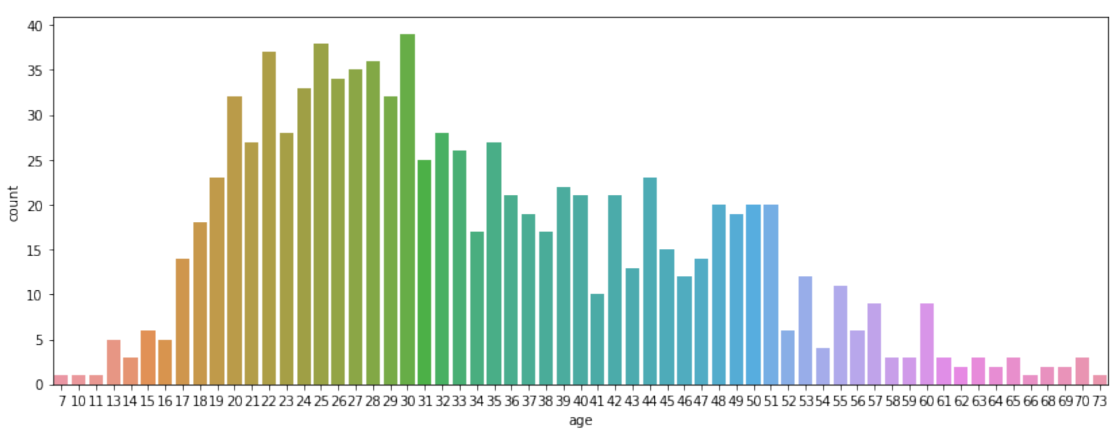
In [8]:
```
sns.countplot(x='rating', data=df_ratings)
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7f709d726550>



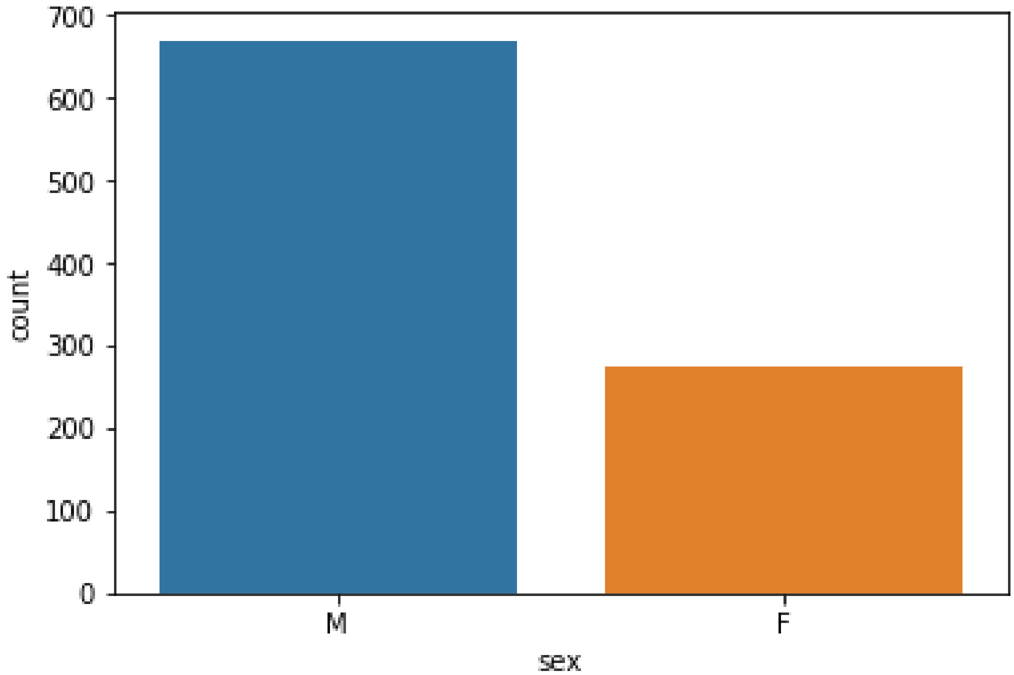
In [9]:
```
plt.figure(figsize=(14, 5))
sns.countplot(x='age', data=df_users)
```

Out[9]: <matplotlib.axes._subplots.AxesSubplot at 0x7f709d726470>



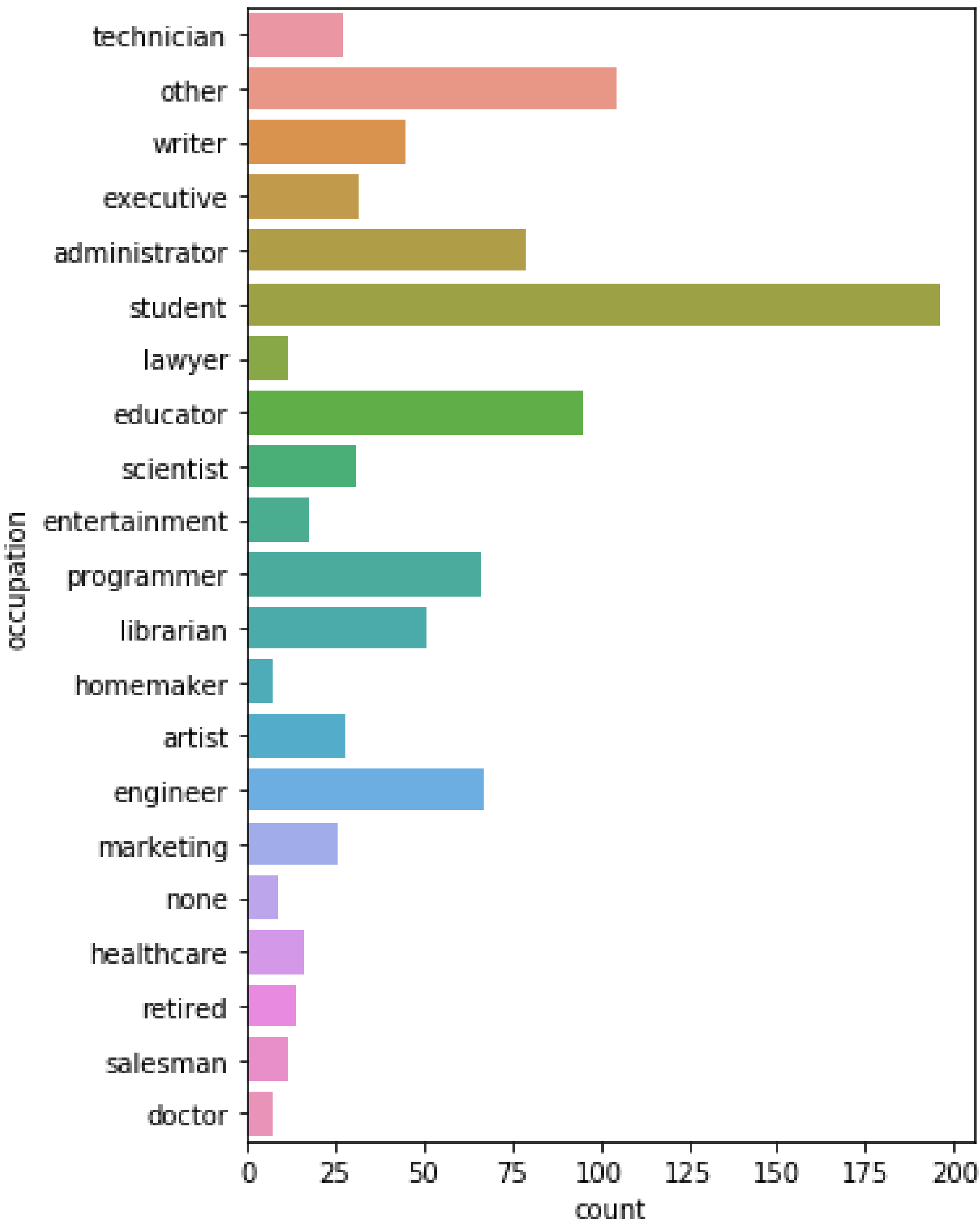
In [10]:
```
sns.countplot(x='sex', data=df_users)
```

Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7f709d069048>



In [11]:
```
plt.figure(figsize=(5, 8))
sns.countplot(y='occupation', data=df_users)
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7f709d237208>

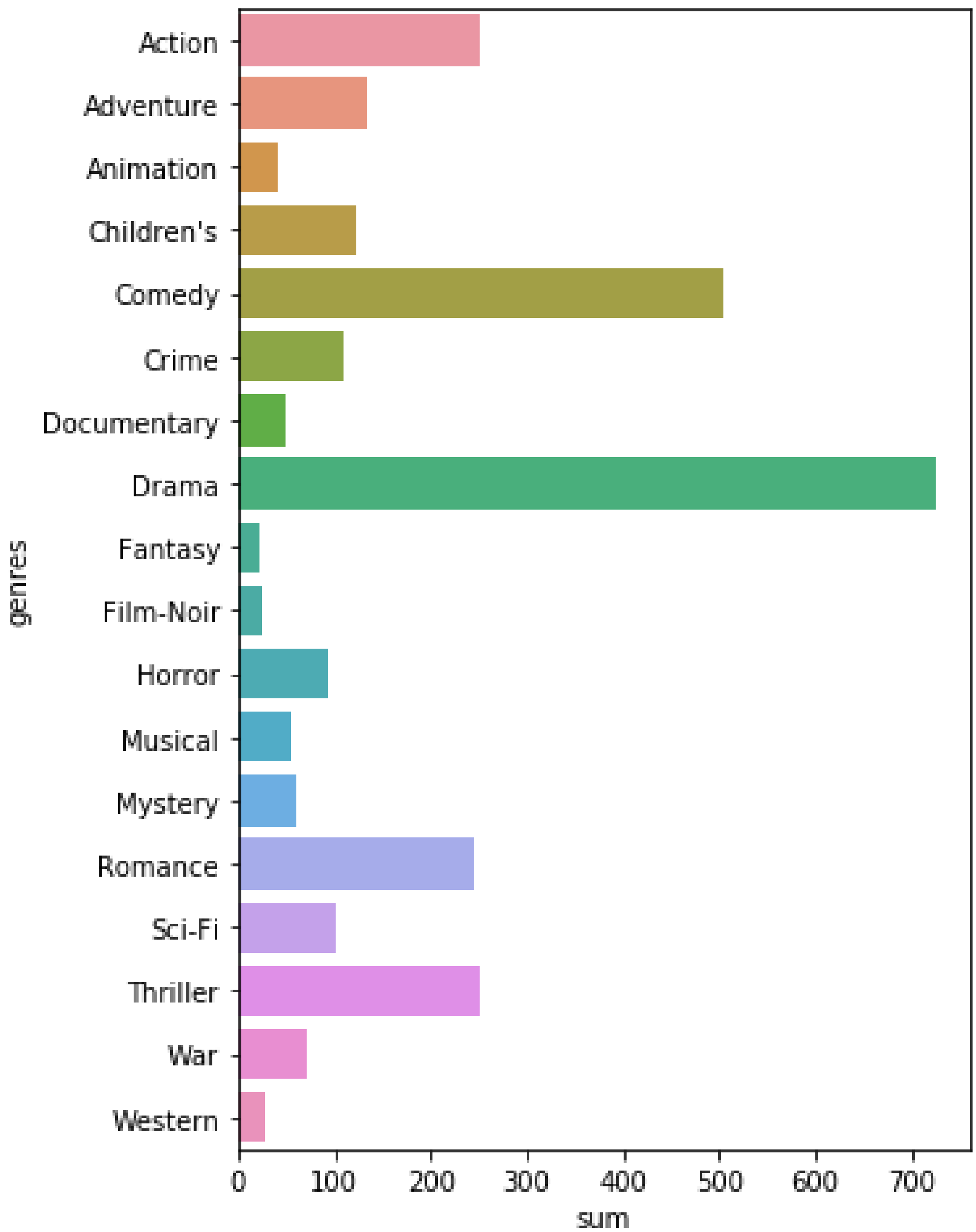



```
In [12]: genre_list = ['Action' , 'Adventure' , 'Animation' , "Children's" , 'Comedy' , 'Crime
                     , 'Fantasy' , 'Film-Noir' , 'Horror' , 'Musical' , 'Mystery' , 'Romance
                     'War' , 'Western']
         genre_sum = []
         for g in genre_list:
             genre_sum.append(df_items[g].sum())

         genre_df = pd.DataFrame({'genres' : genre_list, 'sum' : genre_sum})
```

```
plt.figure(figsize=(5, 8))
sns.barplot(y='genres', x='sum', data=genre_df)
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7f709d1da898>



2.3 Data preparation

In [68]:
```
df_items.columns
```

```
Out[68]: Index(['movie_title', 'release_date', 'video_release_date', 'IMDb_URL',
                'unknown|', 'Action|', 'Adventure|', 'Animation|', 'Children's|',
                'Comedy|', 'Crime|', 'Documentary|', 'Drama|', 'Fantasy|', 'Film-Noir|',
                'Horror|', 'Musical|', 'Mystery|', 'Romance|', 'Sci-Fi|', 'Thriller|',
                'War|', 'Western|'],
               dtype='object')
```

```
In [69]: df_items = df_items.drop(columns=['movie_title', 'release_date', 'video_release_date'
         df_items = df_items.assign(genres=df_items.values.dot(df_items.columns.values))
         df_items = df_items.drop(columns=['unknown|', 'Action|', 'Adventure|', 'Animation|', '
                'Comedy|', 'Crime|', 'Documentary|', 'Drama|', 'Fantasy|', 'Film-Noir|',\
                'Horror|', 'Musical|', 'Mystery|', 'Romance|', 'Sci-Fi|', 'Thriller|',\
                'War|', 'Western|'])

         df_movies = pd.concat([df_movies, df_items], axis=1)
         df_movies = df_movies.drop(columns=['release_date', 'video_release_date', 'imdb_url'])
         df_movies.head()
```

```
Out[69]:                       title                          genres
         movie_id
         1           Toy Story (1995)  Animation|Children's|Comedy|
         2           GoldenEye (1995)    Action|Adventure|Thriller|
         3          Four Rooms (1995)                     Thriller|
         4          Get Shorty (1995)          Action|Comedy|Drama|
         5             Copycat (1995)         Crime|Drama|Thriller|
```

```
In [70]: df_ratings = df_ratings.drop(columns=['unix_timestamp'])
         df_ratings.head()
```

```
Out[70]:    user_id  movie_id  rating
         0      196       242       3
         1      186       302       3
         2       22       377       1
         3      244        51       2
         4      166       346       1
```

In order to fit a LightFM model, the Dataframe should be transformed to a sparse matrix (i.e a big matrice with a lot of zeros or empty values). Pandas' df & Numpy arrays are not suitable for manipulating this kind of data. We need to use Scipy sparse matrices.

By going so, the information of the ids (userId and movieId) will be lost. Then we will only deal with indices (row number and column number). Therefore, the df_to_matrix function also returns dictionaries mapping indexes to ids (ex: uid_to_idx mapping userId to index of the matrix)

```
In [71]: # Reference : https://github.com/EthanRosenthal/rec-a-sketch

         def threshold_interactions_df(df, row_name, col_name, row_min, col_min):
             """Limit interactions df to minimum row and column interactions.
             Parameters
             ----------
```

```
df : DataFrame
    DataFrame which contains a single row for each interaction between
    two entities. Typically, the two entities are a user and an item.
row_name : str
    Name of column in df which corresponds to the eventual row in the
    interactions matrix.
col_name : str
    Name of column in df which corresponds to the eventual column in the
    interactions matrix.
row_min : int
    Minimum number of interactions that the row entity has had with
    distinct column entities.
col_min : int
    Minimum number of interactions that the column entity has had with
    distinct row entities.
Returns
-------
df : DataFrame
    Thresholded version of the input df. Order of rows is not preserved.
Examples
--------
df looks like:
user_id | item_id
=================
  1001  |  2002
  1001  |  2004
  1002  |  2002
thus, row_name = 'user_id', and col_name = 'item_id'
If we were to set row_min = 2 and col_min = 1, then the returned df would
look like
user_id | item_id
=================
  1001  |  2002
  1001  |  2004
"""

n_rows = df[row_name].unique().shape[0]
n_cols = df[col_name].unique().shape[0]
sparsity = float(df.shape[0]) / float(n_rows*n_cols) * 100
print('Starting interactions info')
print('Number of rows: {}'.format(n_rows))
print('Number of cols: {}'.format(n_cols))
print('Sparsity: {:4.3f}%'.format(sparsity))

done = False
while not done:
    starting_shape = df.shape[0]
    col_counts = df.groupby(row_name)[col_name].count()
```

```
        df = df[~df[row_name].isin(col_counts[col_counts < col_min].index.tolist())]
        row_counts = df.groupby(col_name)[row_name].count()
        df = df[~df[col_name].isin(row_counts[row_counts < row_min].index.tolist())]
        ending_shape = df.shape[0]
        if starting_shape == ending_shape:
            done = True

    n_rows = df[row_name].unique().shape[0]
    n_cols = df[col_name].unique().shape[0]
    sparsity = float(df.shape[0]) / float(n_rows*n_cols) * 100
    print('Ending interactions info')
    print('Number of rows: {}'.format(n_rows))
    print('Number of columns: {}'.format(n_cols))
    print('Sparsity: {:4.3f}%'.format(sparsity))
    return df

def get_df_mappings(df, row_name, col_name):
    """Map entities in interactions df to row and column indices
    Parameters
    ----------
    df : DataFrame
        Interactions DataFrame.
    row_name : str
        Name of column in df which contains row entities.
    col_name : str
        Name of column in df which contains column entities.
    Returns
    -------
    rid_to_idx : dict
        Maps row ID's to the row index in the eventual interactions matrix.
    idx_to_rid : dict
        Reverse of rid_to_idx. Maps row index to row ID.
    cid_to_idx : dict
        Same as rid_to_idx but for column ID's
    idx_to_cid : dict
    """


    # Create mappings
    rid_to_idx = {}
    idx_to_rid = {}
    for (idx, rid) in enumerate(df[row_name].unique().tolist()):
        rid_to_idx[rid] = idx
        idx_to_rid[idx] = rid

    cid_to_idx = {}
    idx_to_cid = {}
    for (idx, cid) in enumerate(df[col_name].unique().tolist()):
```

```
            cid_to_idx[cid] = idx
            idx_to_cid[idx] = cid

    return rid_to_idx, idx_to_rid, cid_to_idx, idx_to_cid


def df_to_matrix(df, row_name, col_name):
    """Take interactions dataframe and convert to a sparse matrix
    Parameters
    ----------
    df : DataFrame
    row_name : str
    col_name : str
    Returns
    -------
    interactions : sparse csr matrix
    rid_to_idx : dict
    idx_to_rid : dict
    cid_to_idx : dict
    idx_to_cid : dict
    """
    rid_to_idx, idx_to_rid,\
        cid_to_idx, idx_to_cid = get_df_mappings(df, row_name, col_name)

    def map_ids(row, mapper):
        return mapper[row]

    I = df[row_name].apply(map_ids, args=[rid_to_idx]).values
    J = df[col_name].apply(map_ids, args=[cid_to_idx]).values
    V = np.ones(I.shape[0])
    interactions = sparse.coo_matrix((V, (I, J)), dtype=np.float64)
    interactions = interactions.tocsr()
    return interactions, rid_to_idx, idx_to_rid, cid_to_idx, idx_to_cid
```

In [73]:
```
ratings_matrix, user_id_to_idx, idx_to_user_id, movie_id_to_idx, idx_to_movie_id = df_
        (df_ratings, row_name='user_id', col_name='movie_id')

ratings_matrix.toarray()
```

Out[73]:
```
array([[1., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

This leads to the creation of 5 new variables:

- a final sparse matrix ratings_matrix (this will be the data used to train the model) and the following utils mappers:
- uid_to_idx
- idx_to_uid
- mid_to_idx
- idx_to_mid

How to use those mappers ?

In [76]:
```
# for instance what movies did the userId 4 rateű
movies_user4 = ratings_matrix.toarray()[user_id_to_idx[4], :]
movies_id_user4 = np.sort(np.vectorize(idx_to_movie_id.get)(np.argwhere(movies_user4>
df_movies.loc[movies_id_user4, :]['title']
```

Out[76]:
```
movie_id
11                             Seven (Se7en) (1995)
50                                 Star Wars (1977)
210    Indiana Jones and the Last Crusade (1989)
258                                  Contact (1997)
260                            Event Horizon (1997)
264                                    Mimic (1997)
271                        Starship Troopers (1997)
288                                   Scream (1996)
294                               Liar Liar (1997)
300                            Air Force One (1997)
301                                 In & Out (1997)
303                              Ulee's Gold (1997)
324                             Lost Highway (1997)
327                                 Cop Land (1997)
328                        Conspiracy Theory (1997)
329                       Desperate Measures (1998)
354                      Wedding Singer, The (1998)
356                              Client, The (1994)
357        One Flew Over the Cuckoo's Nest (1975)
358                                    Spawn (1997)
359                          Assignment, The (1997)
360                               Wonderland (1997)
361                                Incognito (1997)
362                      Blues Brothers 2000 (1998)
Name: title, dtype: object
```

In [78]:
```
# On the other side, what is the value of ratings_matrix for: userId = 4
movieId_list = [11, 50, 210, 324, 8, 9, 10]

movieId_idx = [movie_id_to_idx[i] for i in movieId_list]
movieId_idx

ratings_user4 = ratings_matrix.toarray()[user_id_to_idx[4], movieId_idx]
```

```
ratings_user4
#the values in ratings_matrix tells if a user a rated a movie but don't explicit the
```

Out[78]: array([1., 1., 1., 1., 0., 0., 0.])

---

3 Recommendation model

```
In [80]: from lightfm.cross_validation import random_train_test_split
         from lightfm import LightFM
         from lightfm.evaluation import precision_at_k
```

3.1 Introduction to lightFM

LightFM is a Python implementation of a number of popular recommendation algorithms for both implicit and explicit feedback.

It also makes it possible to incorporate both item and user metadata into the traditional matrix factorization algorithms. It represents each user and item as the sum of the latent representations of their features, thus allowing recommendations to generalise to new items (via item features) and to new users (via user features).

Data expected by the LightFM fit method

From the doc, chapter 'usage':

model = LightFM(no_components=30)

Assuming train is a (no_users, no_items) sparse matrix (with 1s denoting positive, and -1s negative interactions),

you can fit a traditional matrix factorization model by calling:

model.fit(train, epochs=20)

This will train a traditional MF model, as no user or item features have been supplied.

3.2 Splitting data

The dataset is slightly different from what we have been used to with Scikit-Learn (X as features, y as target).

Lightfm provides a random_train_test_split located into cross_validation dedicated to this use-case.

Let's split the data randomly into a train matrix and a test matrix with 20% of interactions into the test set.

```
In [81]: train, test = random_train_test_split(ratings_matrix, test_percentage=0.2)
```

3.3 Metric and model performance evaluation

- The optimized metric is the percentage of top k items in the ranking the user has actually interacted with - i.e how good the ranking produced by the model is.
- We'll evaluate the recommendation engine with the WARP: the Weighted Approximate-Rank Pairwise loss. It maximises the rank of positive examples by repeatedly sampling negative examples until rank violating one is found. Useful when only positive interactions are present and optimising the top of the recommendation list (precision@k) is desired.

3.4 Model training and precision

```
In [82]: model = LightFM(loss='warp-kos', no_components=40, k=3, learning_rate=0.03)
         model.fit(train, epochs=100)
```

```
Out[82]: <lightfm.lightfm.LightFM at 0x7f70d44b97b8>
```

```
In [83]: print("Train precision: %.2f" % precision_at_k(model, train, k=5).mean())
         print("Test precision: %.2f" % precision_at_k(model, test, train_interactions=train, k
```

```
Train precision: 0.91
Test precision: 0.36
```

What does the attribute item_embeddings of model contains?

```
In [84]: ratings_matrix.toarray().shape
```

```
Out[84]: (943, 1682)
```

```
In [85]: # equivalent of the Q matrix with no_components = 10, ie the nb of embeddings / featu
         model.item_embeddings.shape
```

```
Out[85]: (1682, 40)
```

```
In [86]: # equivalent of the P matrix with no_components = 10, ie the nb of embeddings / featu
         model.user_embeddings.shape
```

```
Out[86]: (943, 40)
```

3.5 Similarity scores between pairs of movies

Previously, we've trained a model that factorized our ratings matrix into a U matrix of shape (n_users, no_components) : model.user_embeddings ; and V matrix of shape (n_movies, no_components) : model.item_embeddings).

Now we would like to compute similarity between each pair of movies. To calculate a similarity distance, there are 2 choices * cosine_similarity function * from sklearn.metrics.pairwise import cosine_similarity * cosine_similarity(X, Y) * or pearson_similarity: * import numpy as np * np.corrcoef(X, Y)

```
In [87]: #Compute the similarity_scores of size (n_movies, n_movies), containing for each elem
         from sklearn.metrics.pairwise import cosine_similarity

         similarity_scores = cosine_similarity(model.item_embeddings, model.item_embeddings)
         similarity_scores
```

```
Out[87]: array([[ 0.9999999 ,  0.3090872 , -0.3577202 , ...,  0.04141378,
                  0.10872173,  0.04886636],
                [ 0.3090872 ,  1.        , -0.53334486, ..., -0.2657429 ,
                 -0.18552116, -0.18115026],
```

```
       [-0.3577202 , -0.53334486,  1.        , ...,  0.48775673,
         0.33573252,  0.3887553 ],
       ...,
       [ 0.04141378, -0.2657429 ,  0.48775673, ...,  1.0000001 ,
         0.8434785 ,  0.90591186],
       [ 0.10872173, -0.18552116,  0.33573252, ...,  0.8434785 ,
         1.0000001 ,  0.86416286],
       [ 0.04886636, -0.18115026,  0.3887553 , ...,  0.90591186,
         0.86416286,  1.        ]], dtype=float32)
```

In [88]:
```
# it's the similarity with the "features" for all movies
cosine_similarity(model.item_embeddings, model.item_embeddings).shape
```

Out[88]:
```
(1682, 1682)
```

In [89]:
```
np.corrcoef(model.item_embeddings)
```

Out[89]:
```
array([[ 1.        ,  0.30896349, -0.36367854, ...,  0.04217422,
         0.10564704,  0.0499952 ],
       [ 0.30896349,  1.        , -0.53797214, ..., -0.26560836,
        -0.18870322, -0.18094361],
       [-0.36367854, -0.53797214,  1.        , ...,  0.49398712,
         0.32494301,  0.39578214],
       ...,
       [ 0.04217422, -0.26560836,  0.49398712, ...,  1.        ,
         0.85572397,  0.90588932],
       [ 0.10564704, -0.18870322,  0.32494301, ...,  0.85572397,
         1.        ,  0.87858968],
       [ 0.0499952 , -0.18094361,  0.39578214, ...,  0.90588932,
         0.87858968,  1.        ]])
```

In [90]:
```
np.corrcoef(model.item_embeddings).shape
```

Out[90]:
```
(1682, 1682)
```

---

4 Recommandation engine practical use

In [91]:
```
# For instance what are the 10 most similar movies to movie of idx 21 ?
df_movies.loc[np.vectorize(idx_to_movie_id.get)(similarity_scores[21].argsort()[::-1]
```

Out[91]:

movie_id	title	genres
1568	Vermont Is For Lovers (1992)	Comedy|Romance|
1054	Mr. Wrong (1996)	Comedy|
785	Only You (1994)	Comedy|Romance|
868	Hearts and Minds (1996)	Drama|
852	Bloody Child, The (1996)	Drama|Thriller|

```
1315      Inventing the Abbotts (1997)      Drama|Romance|
703             Widows' Peak (1994)                 Drama|
1452        Lady of Burlesque (1943)      Comedy|Mystery|
847        Looking for Richard (1996)  Documentary|Drama|
1343                Lotto Land (1995)                 Drama|
```

On the assumption that a user likes Scream (movie_id = 288), what would be other recommend movies ? (i.e which movies are the most similar)

Retrieve the top 10 recommendations.

In [92]:
```
#similarity_scores works with idx and you have the movie_id associated to your movie.
df_movies.loc[np.vectorize(idx_to_movie_id.get)(similarity_scores[movie_id_to_idx[288]
```

Out[92]:

movie_id	title	genres
333	Game, The (1997)	Mystery|Thriller|
294	Liar Liar (1997)	Comedy|
328	Conspiracy Theory (1997)	Action|Mystery|Romance|Thriller|
475	Trainspotting (1996)	Drama|
307	Devil's Advocate, The (1997)	Crime|Horror|Mystery|Thriller|
147	Long Kiss Goodnight, The (1996)	Action|Thriller|
245	Devil's Own, The (1997)	Action|Drama|Thriller|War|
300	Air Force One (1997)	Action|Thriller|
156	Reservoir Dogs (1992)	Crime|Thriller|
282	Time to Kill, A (1996)	Drama|

In [ ]: