Recommendation System Project

Problem Statement

For this project I will be making movie recommendations based on the MovieLens (https://grouplens.org/datasets/movielens/latest/ (https://grouplens.org/datasets/movielens/latest/)) dataset from the GroupLens research lab at the University of Minnesota. I will be using the "small" dataset containing 100,000 user ratings.

In this project, my main goal is to build a model that provides top 5 movie recommendations to a user, based on their ratings of other movies. Also, my recommendation system will be using collaborative filtering as the primary mechanism and content-based filtering(technique that uses similarities in features to make decisions) to address the cold start problem.

Recommendation engines that run on collaborative filtering recommend each item based on user actions. The more user actions an item has, the easier it is to tell which user would be interested in it and what other items are similar to it. As time progresses, the system will be able to give more and more accurate recommendations.

Cold start problem simply means that a recommendation engine meets a new visitor for the first time. Because there is no history about a user, the system doesn't know the personal preferences of the user. Getting to know the users is crucial in creating a great user experience for them.

My project consists of 2 main sections:

1. Section 1 includes cold start developing and building a recommender system from scratch.
2. In Section 2 I used Surprise library to build the recommender system.

Reading and Exploring the data

In [1]:
```
# Data manipulation
import pandas as pd
import numpy as np

# Data visualization
import seaborn as sns
import matplotlib.pyplot as plt
from wordcloud import WordCloud, STOPWORDS
%matplotlib inline
sns.set_style("darkgrid")

# Building recommender systems
import surprise
from surprise import Reader, BaselineOnly, KNNBasic, SVD, SVDpp, NMF
from surprise import Dataset
from surprise.model_selection import cross_validate

# Others
from scipy.stats import pearsonr
from tqdm.auto import tqdm

```

In [2]:
```
# import the dataframes
links = pd.read_csv("ml-latest-small/links.csv")
movies = pd.read_csv("ml-latest-small/movies.csv")
ratings = pd.read_csv("ml-latest-small/ratings.csv")
tags = pd.read_csv("ml-latest-small/tags.csv")
```

In [3]:
```
# Display the first 5 entries in each dataframe
display(links.head())
display(movies.head())
display(ratings.head())
display(tags.head())
```

	movieId	imdbId	tmdbId
0	1	114709	862.0
1	2	113497	8844.0
2	3	113228	15602.0
3	4	114885	31357.0
4	5	113041	11862.0

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

	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

	userId	movieId	tag	timestamp
0	2	60756	funny	1445714994
1	2	60756	Highly quotable	1445714996
2	2	60756	will ferrell	1445714992
3	2	89774	Boxing story	1445715207
4	2	89774	MMA	1445715200

In [4]:
```
# Determine and display the min and max ratings received
min_rating = ratings['rating'].min()
max_rating = ratings['rating'].max()
print('Lowest rating: {}'.format(min_rating))
print('Highest rating: {}'.format(max_rating))
```

```
Lowest rating: 0.5
Highest rating: 5.0
```

> Movies are rated between 0 and 5 with the lowest rating being 0.5 and the highest 5.

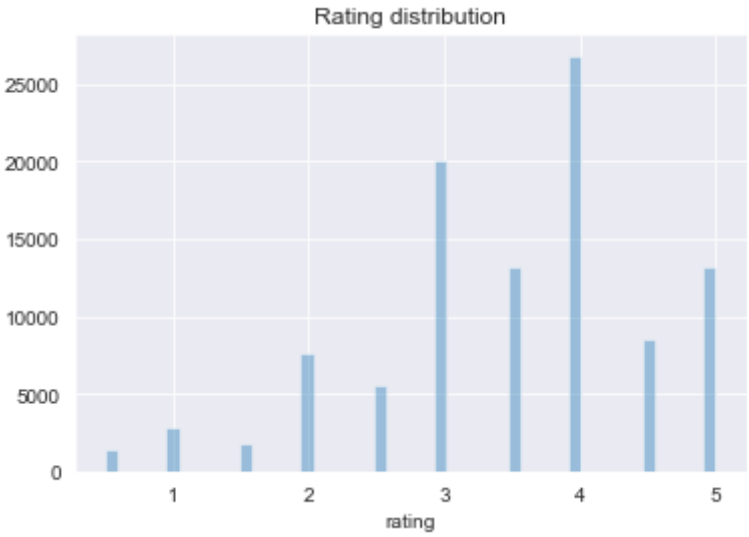
In [5]:
```
ratings.describe()
```

Out[5]:

	userId	movieId	rating	timestamp
count	100836.000000	100836.000000	100836.000000	1.008360e+05
mean	326.127564	19435.295718	3.501557	1.205946e+09
std	182.618491	35530.987199	1.042529	2.162610e+08
min	1.000000	1.000000	0.500000	8.281246e+08
25%	177.000000	1199.000000	3.000000	1.019124e+09
50%	325.000000	2991.000000	3.500000	1.186087e+09
75%	477.000000	8122.000000	4.000000	1.435994e+09
max	610.000000	193609.000000	5.000000	1.537799e+09

> The average rating across all users and movies is 3.5.

In [6]:
```
sns.distplot(ratings['rating'], kde = False);
plt.title('Rating distribution')
plt.show()
```



> Most of the movies have less than 25000 ratings. Also, most of the users rated movies with 4.0.

In [7]:
```
# is any row null in links
links.isnull().any()
```

Out[7]:
```
movieId    False
imdbId     False
tmdbId      True
dtype: bool
```

In [8]:
```

# lets drop null rows
links = links.dropna()
```

In [9]:
```

# is any row has null
movies.isnull().any()
```

Out[9]:
```
movieId    False
title      False
genres     False
dtype: bool
```

In [10]:
```
# check ratings
ratings.isnull().any()
```

Out[10]:
```
userId       False
movieId      False
rating       False
timestamp    False
dtype: bool
```

In [11]:
```
# check for tags
tags.isnull().any()
```

Out[11]:
```
userId       False
movieId      False
tag          False
timestamp    False
dtype: bool
```

Explore `tags` to see if there is any relevant information for my project

In [12]:
```
# Define a function to plot word cloud
def plot_cloud(wordcloud):
    plt.figure(figsize =(10, 8))
    plt.imshow(wordcloud)
    plt.axis("off");



wordcloud = WordCloud(width = 1000,
                      height = 500,
                      random_state = 1,
                      background_color ='salmon',
                      colormap ='Pastel1',
                      collocations = True,
                      stopwords = STOPWORDS).generate(' '.join(tags['ta
plot_cloud(wordcloud)
```

Data in `tags` is mixed it contains streaming service name(Netflix), genres, random words, countries. Because these information is irrelevant for my research I am not going to use it.

Data preprocessing

To achieve my project goals, I will be using movie genres data but first I have to prepare it.

In [13]:
```
#  movie: Adventure|Children|Fantasy  --> 1 0 1 1 0 0
movies = pd.read_csv("ml-latest-small/movies.csv")
all_genres = set()

#  define all genres that can be found in the "genres" column
for movie_genres in movies['genres']:
    all_genres = all_genres | set(movie_genres.split('|'))  # merge two

#  add new columns to the dataframe and fill them with zeros
for g in all_genres:
    movies[g] = 0

years = []
movies['year'] = 0
#  put 1's in right places
all_titles = ''
for i in tqdm(range(len(movies['genres']))):
    movie_genres = movies['genres'][i]
    title = movies['title'][i]
    try:
        movies['year'][i] = int(title.strip()[-5:-1])
        movies['title'][i] = title[:-6].strip()
        all_titles += ' ' + movies['title'][i] + ' '
    except:
        continue

    for g in all_genres:
        if g in set(movie_genres.split('|')):
            movies[g][i] = 1


all_genres = list(all_genres)
movies = movies.set_index('movieId')
movies.head()
```

```
100%                                  9742/9742 [00:06<00:00, 1407.11it/s]

/Users/alinakorsunenko/opt/anaconda3/envs/learn-env/lib/python3.6/site-pa
ckages/ipykernel_launcher.py:21: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy)
/Users/alinakorsunenko/opt/anaconda3/envs/learn-env/lib/python3.6/site-pa
ckages/ipykernel_launcher.py:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http://
pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a
-view-versus-a-copy)
/Users/alinakorsunenko/opt/anaconda3/envs/learn-env/lib/python3.6/site-pa
ckages/ipykernel_launcher.py:29: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame
```

```
See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
```

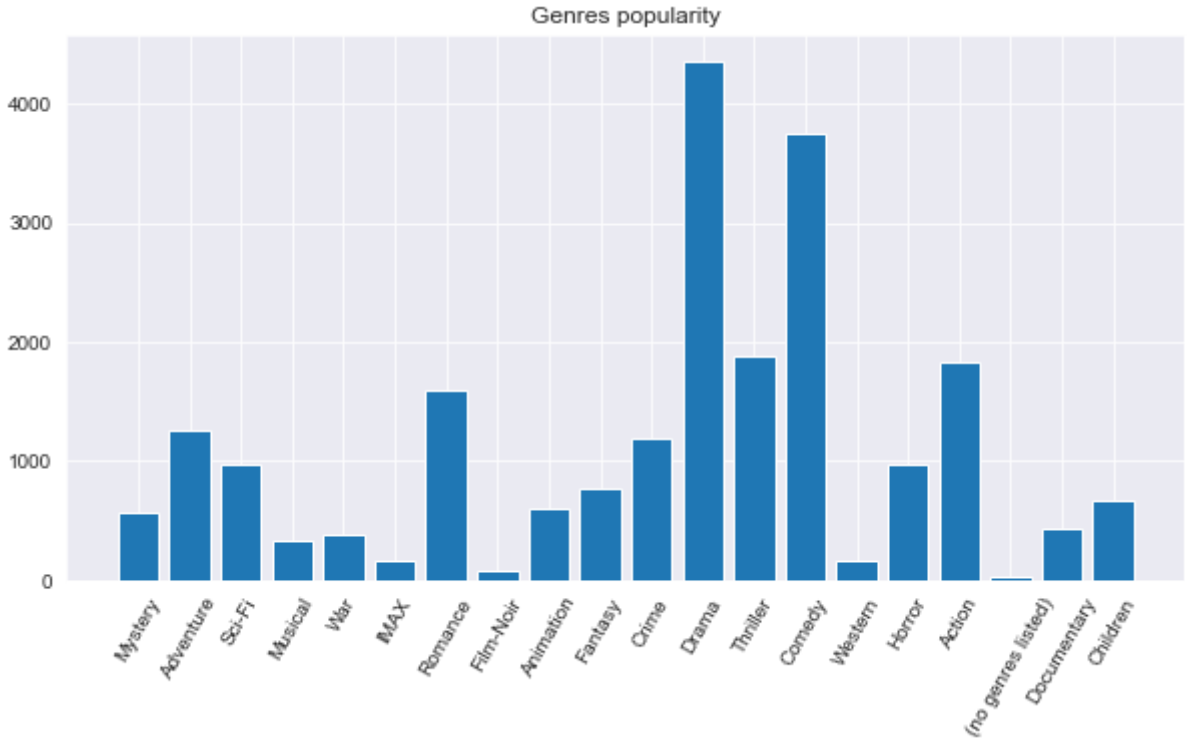
Out[13]:

movieId	title	genres	Mystery	Adventure	Sci-Fi	Musical	W
1	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	0	1	0	0	
2	Jumanji	Adventure|Children|Fantasy	0	1	0	0	
3	Grumpier Old Men	Comedy|Romance	0	0	0	0	
4	Waiting to Exhale	Comedy|Drama|Romance	0	0	0	0	
5	Father of the Bride Part II	Comedy	0	0	0	0	

5 rows × 23 columns

> 1 indicates that a movie belogs to a certain movie genre and 0 that it does not.

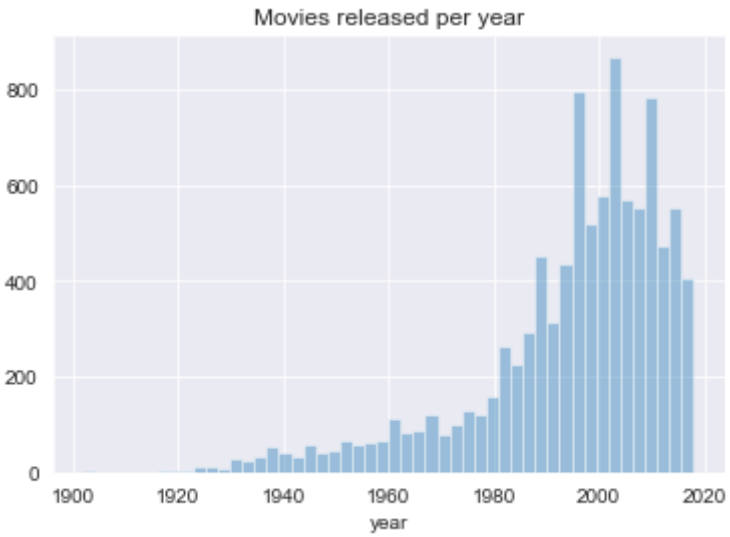
In [14]:
```
# explore genres and how are they distributed.
plt.figure(figsize = (10, 5))
genres_stat = movies[all_genres].values.sum(axis = 0)
plt.bar(all_genres, genres_stat)
plt.xticks(rotation = 60)
plt.title("Genres popularity")
plt.show()
```



Most of the movies fall into the drama, comedy and thriller genres.

In [15]:
```
# distribution of movie releases per year
sns.distplot(movies['year'][movies['year'] != 0], kde = False);
plt.title('Movies released per year')
plt.show()

```



movie production started growing between 1990 and 2000 and then slowed down closer to 2020.

In [16]:
```
# Explore movie titles by defining a function to plot word cloud
def plot_cloud(wordcloud):
    plt.figure(figsize =(10, 8))
    plt.imshow(wordcloud)
    plt.axis("off");

# Generate word cloud
wordcloud = WordCloud(width = 1000,
                      height = 500,
                      random_state = 1,
                      background_color = 'salmon',
                      colormap = 'Pastel1',
                      collocations = True,
                      stopwords = STOPWORDS).generate(all_titles)
plot_cloud(wordcloud)
```

Section 1

First part of this section consists of - `Cold start developing` where I will be defining a function which returns 5 best films(cold start problem happens when new users or new items arrive in e-commerce platforms and it makes it difficult to detact personal preferences of the user).Eventually, I want to have a list of movies which have both high average ratings and high number of ratings. I will start with merging two datasets to find the average ratings and the number of ratings for a movie and then by looking at distribution of users rating decide if I have to define a threshold.

In the second part of this section - `Building recommender system` I will be building a simple recommender system which will focus on movie genres which are the most similar to those viewed by the users.In case if there will not be any information about a movie genre my recommender system will use a cold start.

In the third part of this section - `Comparison` , I will be comparing results of the recommender system and the cold start but to do that I will build a metric by defining a function which will return a number of movies in recommendations that user watched before.

Cold start developing

In [17]:
```
# merging two dataframes "movies.csv" and "ratings.csv"
movie_data = pd.merge(ratings, movies, on = 'movieId')
movie_data.head()
```

Out[17]:

	userId	movieId	rating	timestamp	title	genres	Mystery	A
0	1	1	4.0	964982703	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	0	
1	5	1	4.0	847434962	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	0	
2	7	1	4.5	1106635946	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	0	
3	15	1	2.5	1510577970	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	0	
4	17	1	4.5	1305696483	Toy Story	Adventure|Animation|Children|Comedy|Fantasy	0	

5 rows × 27 columns

In [18]:
```
# find out the average rating for each movie in the dataset.
movie_data.groupby('title')['rating'].mean().head()
```

Out[18]:
```
title
'71                                     4.0
'Hellboy': The Seeds of Creation        4.0
'Round Midnight                         3.5
'Salem's Lot                            5.0
'Til There Was You                      4.0
Name: rating, dtype: float64
```

In [19]:
```
# sort the ratings in the ascending order of their average ratings:
movie_data.groupby('title')['rating'].mean().sort_values(ascending = Fa
```

Out[19]:
```
title
Sorority House Massacre                           5.0
Entertaining Angels: The Dorothy Day Story        5.0
Passenger, The (Professione: reporter)            5.0
Little Dieter Needs to Fly                        5.0
Human Condition III, The (Ningen no joken III)    5.0
Name: rating, dtype: float64
```

A movie can make it to the top of the above list even if only a single user has given it five stars. Therefore, the above results can be misleading. Normally, a movie which is really a good one gets a higher rating by a large number of users.

In [20]:
```
# plot the total number of ratings for a movie:
movie_data.groupby('title')['rating'].count().sort_values(ascending = F
```

Out[20]:
```
title
Forrest Gump                  329
Shawshank Redemption, The     317
Pulp Fiction                  307
Silence of the Lambs, The     279
Matrix, The                   278
Name: rating, dtype: int64
```

The above list supports the point that good movies normally receive higher ratings.

In [21]:
```
# add the average rating of each movie to ratings_mean_count dataframe
ratings_mean_count = pd.DataFrame(movie_data.groupby('title')[['rating'
```

In [22]:
```
# add the number of ratings for a movie to the ratings_mean_count dataf
ratings_mean_count['rating_counts'] = pd.DataFrame(movie_data.groupby('
```

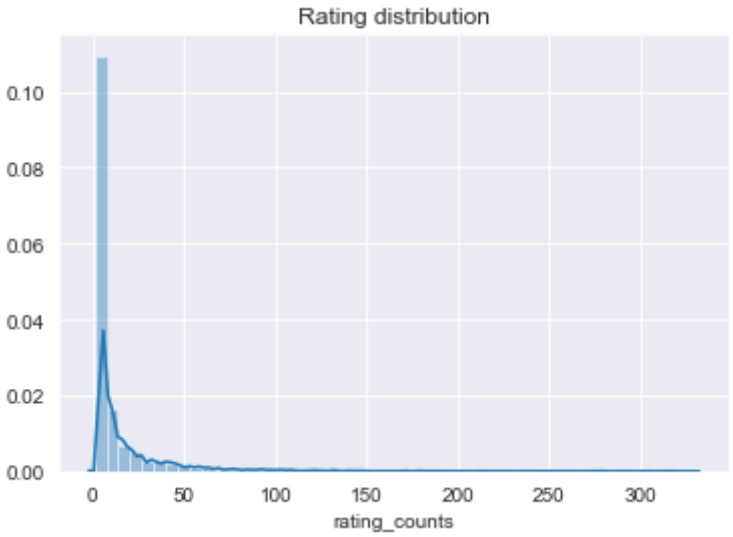
In [23]:
```
# convert movieId to int
ratings_mean_count['movieId'] = ratings_mean_count['movieId'].astype(in
ratings_mean_count.head()
```

Out[23]:

title	rating	movieId	rating_counts
'71	4.0	117867	1
'Hellboy': The Seeds of Creation	4.0	97757	1
'Round Midnight	3.5	26564	2
'Salem's Lot	5.0	27751	1
'Til There Was You	4.0	779	2

> Now I can see movie title, along with the average rating and number of ratings for the movie.

In [24]:
```
# plot distribution of users rating
sns.distplot(ratings_mean_count['rating_counts']);
plt.title('Rating distribution')
plt.show()
```



From the output, I can conclude that most of the movies have received less than 50 ratings. While the number of movies having more than 100 ratings is very low. I want to exclude movies with high ratings and small number of and movies with a lot of low ratings, tnerefore, I will define a threshold.

In [25]:
```
# define a threshold
q = ratings_mean_count['rating_counts'].quantile(0.95)  # 47

# filter ratings
ratings_mean_count_filtered = ratings_mean_count[ratings_mean_count['ra
```

In [26]:
```
# check results
ratings_mean_count_filtered.sort_values('rating', ascending = False).he
```

Out[26]:

title	rating	movieId	rating_counts
Shawshank Redemption, The	4.429022	318	317
Godfather, The	4.289062	858	192
Fight Club	4.272936	2959	218
Cool Hand Luke	4.271930	1276	57
Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb	4.268041	750	97
Rear Window	4.261905	904	84
Godfather: Part II, The	4.259690	1221	129
Departed, The	4.252336	48516	107
Goodfellas	4.250000	1213	126
Casablanca	4.240000	912	100

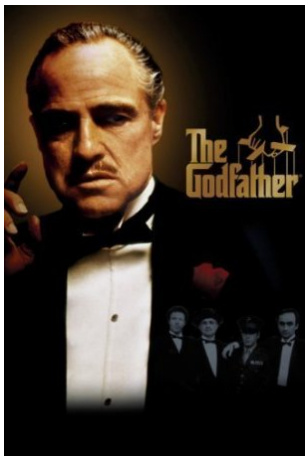
An interesting thing to note is that all the movies in the top 10 are older, this just could be because these movies have been around longer and have been rated more as a result

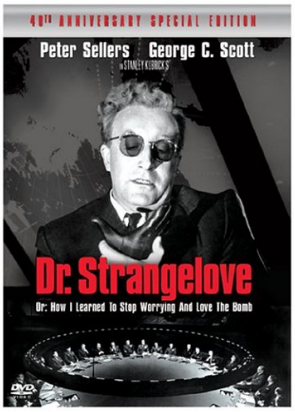
In [27]:
```
# define best films(movie ids) which both popular and have high ratings
best_films_ids = ratings_mean_count_filtered.sort_values('rating', asce

# define a function which returns 5 best films(movie ids) based on aver
def cold_start(user, ratings, movies):

    return best_films_ids

```





Building the recommender system

In [28]:
```
def recommend(userId, ratings, movies): # -> List[int]:
    user_favorites = ratings[ratings['userId'] == userId].sort_values('
    #  user_favorites contains id's of user's favourite movies
    answer = []

    rec_number = 5

    for movie_id in user_favorites:
        try:
            movie_descr = movies[all_genres].iloc[movie_id]  #  descrip
        except:
            return cold_start(userId, ratings, movies)

        top_sim = dict() # {sim: index}

        lowest_top_sim = 0
        lowest_top_idx = 0

        for id, curr_description in movies[all_genres].iterrows():
            similarity, _ = pearsonr(movie_descr, curr_description)

            if len(top_sim) < rec_number:
                top_sim[similarity] = id
                continue

            #  update current top and check if we haven't added this fi
            min_top = min(list(top_sim.keys()))
            if similarity > min_top and id != movie_id and id not in an
                del top_sim[min_top]  # remove min of tops
                top_sim[similarity] = id  # add new
        answer += list(top_sim.items())

    return answer
```

Comparison

In [29]:
```
np.random.seed(179)

# define a function which returns number of movies in recommendations t
def accuracy(ratings, movies, method):
    score = 0
    random_users = np.unique(np.random.choice(ratings['userId'].values,
    for user_id in tqdm(random_users):
        recs = method(user_id, ratings, movies)
        user_films = np.unique(ratings[ratings['userId'] == user_id]['m
        for r in recs:
            if r in user_films:
                score += 1
    return score
```

In [30]:
```
# check results for recommend

accuracy(ratings, movies, recommend)
```

```
100%                                          9/9 [00:52<00:00, 5.86s/it]

/Users/alinakorsunenko/opt/anaconda3/envs/learn-env/lib/python3.6/site-pa
ckages/scipy/stats/stats.py:3845: PearsonRConstantInputWarning: An input
array is constant; the correlation coefficent is not defined.
  warnings.warn(PearsonRConstantInputWarning())
/Users/alinakorsunenko/opt/anaconda3/envs/learn-env/lib/python3.6/site-pa
ckages/ipykernel_launcher.py:11: DeprecationWarning: elementwise comparis
on failed; this will raise an error in the future.
  # This is added back by InteractiveShellApp.init_path()
```

Out[30]: 12

In [31]:
```
# check results for cold start

accuracy(ratings, movies, cold_start)
```

```
100%                                          10/10 [00:00<00:00, 85.41it/s]
```

Out[31]: 27

I can assume that my cold start shows better result (27 movies) than recommend(12 movies) because there is not enough data about users.

Section 2

Surprise package for collaborative filtering.

I will use the Surprise library that provides various ready-to-use powerful prediction algorithms to

evaluate its RMSE (Root Mean Squared Error) and FCP(Fraction of Concordant Pairs) on the MovieLens dataset. In this section I will use 3 models: `KNN` (K nearest neighbors), `SVD` (Singular Value Decomposition) and `NMF` (Non-negative matrix factorization). All these methods have different approaches of performance evaluation and that is why I have decided to use them.

FCP. It is the fraction of concordant pairs among all the pairs (sum over all users). Suppose a user has rated n products, then there are $\frac{n(n-1)}{2}$ unique pairs of ratings. If product A receives a higher rating than product B from a user and the model predicts the same, A and B are a concordant pair, otherwise a discordant pair.

KNN. Helps determine if a data point will be part of one class or another. It looks at the neighboring data points to determine what this new data point will fall into. For example, it is determining if someone will be a republican or democrat based on their income and age.

SVD. Its purpose is to reduce a dataset containing a large number of values to a dataset containing significantly fewer values, but which still contains a large fraction of the variability present in the original data.

NMF. It is useful when there are many attributes and the attributes are ambiguous or have weak predictability. By combining attributes, NMF can produce meaningful patterns, topics, or themes.

In [32]:
```
# Load Reader library
reader = Reader(line_format = 'user item rating timestamp', sep = '\t')

# Load ratings dataset with Dataset library
data = Dataset.load_from_df(ratings[['userId', 'movieId', 'rating']], r

# Evaluating the performance of KNN model
knn_results = cross_validate(KNNBasic(), data, measures=['rmse', 'mae',
np.mean(knn_results['test_rmse'])
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Evaluating RMSE, MAE, FCP of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9520  0.9455  0.9431  0.9498  0.9454  0.9472  0.0032
MAE (testset)     0.7295  0.7258  0.7239  0.7299  0.7226  0.7264  0.0029
FCP (testset)     0.6643  0.6748  0.6711  0.6789  0.6672  0.6713  0.0052
Fit time          0.16    0.15    0.20    0.21    0.19    0.18    0.02
Test time         2.16    2.03    2.11    1.97    2.40    2.13    0.15
```

Out[32]:
```
0.9471558734446839
```

The `RMSE` in the case of KNN is 0.95 and `FCP` is 0.67. Let's check other models and compare results.

In [33]:
```
# Use the SVD algorithm.
svd = SVD()

# Compute the RMSE of the SVD algorithm.
cross_validate(svd, data, measures=['RMSE', 'MAE', 'FCP'], cv = 5, verb
```

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8707  0.8701  0.8829  0.8671  0.8789  0.8739  0.0059
MAE (testset)     0.6684  0.6696  0.6795  0.6663  0.6745  0.6717  0.0048
FCP (testset)     0.6594  0.6500  0.6561  0.6584  0.6668  0.6581  0.0054
Fit time          6.44    6.42    5.92    6.45    6.02    6.25    0.23
Test time         0.26    0.21    0.16    0.16    0.30    0.22    0.06
```

Out[33]:
```
{'test_rmse': array([0.87074974, 0.87005368, 0.88288374, 0.86711355, 0.87
894177]),
 'test_mae': array([0.66843776, 0.66955192, 0.67953432, 0.66626801, 0.674
54017]),
 'test_fcp': array([0.65936828, 0.64996907, 0.65613391, 0.65841139, 0.666
76293]),
 'fit_time': (6.444454193115234,
  6.415081024169922,
  5.923551082611084,
  6.45039701461792,
  6.021747827529907),
 'test_time': (0.26468491554260254,
  0.2128748893737793,
  0.159865140914917,
  0.1625077724456787,
  0.30130791664123535)}
```

> The errors has reduced to `RMSE` value of 0.87 and `FCP` value of 0.66 which is better than results of the previous approach I used.

In [34]:
```
# Use the NMF algorithm.
nmf = NMF()

# Compute the RMSE of the SVD algorithm.
cross_validate(nmf, data, measures=['RMSE', 'MAE', 'FCP'], cv = 5, verb
```

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9155  0.9116  0.9273  0.9205  0.9145  0.9179  0.0055
MAE (testset)     0.7013  0.6952  0.7104  0.7057  0.7019  0.7029  0.0051
FCP (testset)     0.6522  0.6620  0.6474  0.6550  0.6491  0.6531  0.0051
Fit time          7.87    6.96    8.79    6.71    7.33    7.53    0.74
Test time         0.18    0.16    0.24    0.24    0.17    0.20    0.04
```

Out[34]:
```
{'test_rmse': array([0.91550888, 0.91156654, 0.92732041, 0.92045536, 0.91
445058]),
 'test_mae': array([0.70133784, 0.69515217, 0.71041534, 0.70566169, 0.701
86844]),
 'test_fcp': array([0.65222377, 0.66202225, 0.64744254, 0.65496033, 0.649
09984]),
 'fit_time': (7.871050834655762,
  6.958197116851807,
  8.789987087249756,
  6.708987236022949,
  7.3304760456085205),
 'test_time': (0.18143510818481445,
  0.15832090377807617,
  0.24203991889953613,
  0.24292683601379395,
  0.16597294807434082)}
```

> The `RMSE` in the case of `NMF` is 0.92 and `FCP` is 0.65 which is lower than `KNN` results but higher than `SVD`

I evaluated each model with the appropriate evaluation metric. Also, to evaluate collaborative filtering algorithm by using KNN, SVD and NMF as metrics. For SVD model I got an RMSE of 0.87 and FCP of 0.66 which are the best results among others.

Conclusion

The "small"(100,000 user ratings) MovieLens dataset was used to build the recommender system. This dataset contains movie ratings and movie specific data relevant for my research. After doing an exploratory data analysis I noted that a lot of movies were produced during 1990 and 2000. Also,according to the dataset Drama, comedy and thriller are the most popular genres.

I developed a cold start and got a list of top 5 movies based on average ratings given by users : The Shawshank Redemption, The Godfather, The Fight Club, Cool Hand Luke and Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb.Also, I built a recommender but its relsults

were worse than the cold start's.

My last part of the project consists of implementing surprise package for collaborative filtering where I used 3 methods KNN, SVD, and NMF methods to evaluate RMSE and FCP on the MovieLens dataset.

The SVD model shows the best results and was able to predict new ratings with a RMSE of 0.87 and FCP of 0.66.

Future work

1. Improve recommender by using dataset with more information about users (gender, age, nationality...)

In [ ]:
```
1
```