Movie Recommendation System

The goal of this notebook is to create a recommendation system that will give recommendations of movies based on user input.

First look at the data

In [1]:

```
# Importing the datasets
import pandas as pd
import numpy as np
movies = pd.read_csv('ml-latest-small/movies.csv')
ratings = pd.read_csv('ml-latest-small/ratings.csv')
```

In [2]:

```
movies.head()
```

Out[2]:

	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

In [3]:

```
ratings.head()
```

Out[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

I'll be using surprise to do my prediction and modeling so I need to reduce the ratings data down to three columns. Time stamp is extraneous to begin with so I'll drop that column entirely.

In [4]:

```
ratings = ratings.drop(columns = 'timestamp')
ratings.head()
```

Out[4]:

	userId	movieId	rating
0	1	1	4.0
1	1	3	4.0
2	1	6	4.0
3	1	47	5.0
4	1	50	5.0

Perfect. And much less effort thanks in part to not having any data cleaning to do. Now, we need to do a little data model testing.

Model Testing

In [5]:

```
# This is for checking the time it takes to run each individual model
# Running this cell might take a while
import time

from surprise.similarities import cosine, msd, pearson
from surprise import accuracy
from surprise import Reader, Dataset
from surprise.model_selection import train_test_split

#loading the .CSV file into surprise
reader = Reader()
data = Dataset.load_from_df(ratings,reader)
train, test = train_test_split(data, test_size=0.2)

rmseScores = []

from surprise.prediction_algorithms import knns
sim_pearson = {'name':'pearson', 'user_based':False}
basic_pearson = knns.KNNBasic(sim_options=sim_pearson)
start = time.time()
basic_pearson.fit(train)
predictions = basic_pearson.test(test)
thePrediction = f'KNNBasic: {accuracy.rmse(predictions)}'
end = time.time()
store = f'{thePrediction} | Time Elapsed: {np.round(end - start, 2)}/sec'
rmseScores.append(store)

knn_means = knns.KNNWithMeans(sim_options=sim_pearson)
start = time.time()
knn_means.fit(train)
predictions = knn_means.test(test)
thePrediction = f'KNNWithMeans: {accuracy.rmse(predictions)}'
end = time.time()
store = f'{thePrediction} | Time Elapsed: {np.round(end - start, 2)}/sec'
rmseScores.append(store)

knnZ = knns.KNNWithZScore(sim_options=sim_pearson)
start = time.time()
knnZ.fit(train)
predictions = knnZ.test(test)
thePrediction = f'KNNWithZScore: {accuracy.rmse(predictions)}'
end = time.time()
store = f'{thePrediction} | Time Elapsed: {np.round(end - start, 2)}/sec'
rmseScores.append(store)

sim_pearson = {'name':'pearson', 'user_based':False}
knn_baseline = knns.KNNBaseline(sim_options=sim_pearson)
start = time.time()
knn_baseline.fit(train)
predictions = knn_baseline.test(test)
thePrediction = f'KNNBaseline: {accuracy.rmse(predictions)}'
end = time.time()
store = f'{thePrediction} | Time Elapsed: {np.round(end - start, 2)}/sec'
rmseScores.append(store)

from surprise.prediction_algorithms import SVD
svd = SVD()
start = time.time()
svd.fit(train)
predictions = svd.test(test)
thePrediction = f'SVD: {accuracy.rmse(predictions)}'
end = time.time()
store = f'{thePrediction} | Time Elapsed: {np.round(end - start, 2)}/sec'
rmseScores.append(store)

from surprise import NormalPredictor
normPred = NormalPredictor()
start = time.time()
normPred.fit(train)
predictions = normPred.test(test)
thePrediction = f'NormalPredictor: {accuracy.rmse(predictions)}'
end = time.time()
store = f'{thePrediction} | Time Elapsed: {np.round(end - start, 2)}/sec'
rmseScores.append(store)

from surprise import BaselineOnly
baseline = BaselineOnly()
start = time.time()
baseline.fit(train)
```

```python
predictions = baseline.test(test)
thePrediction = f'BaselineOnly: {accuracy.rmse(predictions)}'
end = time.time()
store = f'{thePrediction} | Time Elapsed: {np.round(end - start, 2)}/sec'
rmseScores.append(store)

from surprise.prediction_algorithms import NMF
NMF = NMF()
start = time.time()
NMF.fit(train)
predictions = NMF.test(test)
thePrediction = f'NMF: {accuracy.rmse(predictions)}'
end = time.time()
store = f'{thePrediction} | Time Elapsed: {np.round(end - start, 2)}/sec'
rmseScores.append(store)

from surprise.prediction_algorithms import SlopeOne
slopeOne = SlopeOne()
start = time.time()
slopeOne.fit(train)
predictions = slopeOne.test(test)
thePrediction = f'SlopeOne: {accuracy.rmse(predictions)}'
end = time.time()
store = f'{thePrediction} | Time Elapsed: {np.round(end - start, 2)}/sec'
rmseScores.append(store)

from surprise.prediction_algorithms import CoClustering
cluster = CoClustering()
start = time.time()
cluster.fit(train)
predictions = cluster.test(test)
thePrediction = f'CoClustering: {accuracy.rmse(predictions)}'
end = time.time()
store = f'{thePrediction} | Time Elapsed: {np.round(end - start, 2)}/sec'
rmseScores.append(store)

rmseScores
```

```
Computing the pearson similarity matrix...
Done computing similarity matrix.
RMSE: 0.9717
Computing the pearson similarity matrix...
Done computing similarity matrix.
RMSE: 0.9037
Computing the pearson similarity matrix...
Done computing similarity matrix.
RMSE: 0.9073
Estimating biases using als...
Computing the pearson similarity matrix...
Done computing similarity matrix.
RMSE: 0.8845
RMSE: 0.8817
RMSE: 1.4108
Estimating biases using als...
RMSE: 0.8782
RMSE: 0.9248
RMSE: 0.9015
RMSE: 0.9443
```

Out[5]:

```
['KNNBasic: 0.9717286081189962 | Time Elapsed: 68.85/sec',
 'KNNWithMeans: 0.9037395825350144 | Time Elapsed: 65.91/sec',
 'KNNWithZScore: 0.9072937188920338 | Time Elapsed: 68.44/sec',
 'KNNBaseline: 0.8844565552387728 | Time Elapsed: 72.71/sec',
 'SVD: 0.8817477121399103 | Time Elapsed: 13.71/sec',
 'NormalPredictor: 1.4108249897576761 | Time Elapsed: 0.51/sec',
 'BaselineOnly: 0.8782251683286775 | Time Elapsed: 0.44/sec',
 'NMF: 0.9248299878053662 | Time Elapsed: 13.96/sec',
 'SlopeOne: 0.9014967709001301 | Time Elapsed: 23.55/sec',
 'CoClustering: 0.9442535015507145 | Time Elapsed: 5.98/sec']
```

SVD and BaselineOnly seems to have the closest accuracy while maintaining a very short runtime. Although KNNBaseline has the second best score, all KNN models have a substantial runtime that holds it back. Looking at BaselineOnly, modifying the parameters seems to be missing some documentation.

Let's commit to SVD and run a grid search to narrow in what parameters might work best.

In [6]:

```python
# Importing relevant libraries here just to get them out of the way
```

```
from surprise.model_selection import cross_validate
from surprise.model_selection import GridSearchCV
```

In [7]:

```
# Parameters for the grid search and setting them
params = {'n_factors': [20, 50, 100], 'reg_all': [0.02, 0.05, 0.1]}
GSsvd = GridSearchCV(SVD, param_grid = params, n_jobs = -1)
```

In [8]:

```
#fitting it on the data
GSsvd.fit(data)
```

In [9]:

```
# print out optimal parameters for SVD after GridSearchbest_params
print(GSsvd.best_score)
print(GSsvd.best_params)
```

```
{'rmse': 0.8689998723735289, 'mae': 0.6680109289584346}
{'rmse': {'n_factors': 50, 'reg_all': 0.05}, 'mae': {'n_factors': 50, 'reg_all': 0.05}}
```

Perfect. Finally, let's build one solid function that takes in some input, runs our model, spits out some recommendations of movies.

In [10]:

```
# This function will take in the predictions and give the top recommendations.
# This function is separate because it'll be called in the next fucntion.
def recommended_movies(user_ratings,movie_title_df,n):
        for idx, rec in enumerate(user_ratings):
            title = movie_title_df.loc[movie_title_df['movieId'] == int(rec[0])]['title']
            print('Recommendation # ', idx+1, ': ', title, '\n')
            n -= 1
            if n == 0:
                break
```

In [14]:

```
def movie_recommender(movie_df, num_of_rated_movies, genre=None):
    userID = 1000
    rating_list = []
    print(f'Thank you for participating! In order to obtain your recommendations, please rate {num_of_rate
d_movies} movies.')

    # This portion grabs a random movie title and info and asks the user to rate it
    # Once the user gives a number 1-5 or n (for haven't seen), it will append it
    # To the rating_list.
    while num_of_rated_movies > 0:
        if genre:
            movie = movie_df[movie_df['genres'].str.contains(genre)].sample(1)
        else:
            movie = movie_df.sample(1)
        print(movie)
        rating = input('On a scale of 1 - 5, how would you rate this movie? press n if you have not seen
this movie. Press enter to submit your answer: \n')
        if rating == 'n':
            continue
        else:
            rating_one_movie = {'userId':userID, 'movieId':movie['movieId'].values[0], 'rating':rating}
            rating_list.append(rating_one_movie)
            num_of_rated_movies -= 1

    # This portion will take the ratings list, and makes a prediction on it
    new_rating_df = ratings.append(rating_list, ignore_index = True)
    new_data = Dataset.load_from_df(new_rating_df, reader)
    svd = SVD(n_factors=100, n_epochs=10, lr_all=0.005, reg_all=0.4)
    svd.fit(new_data.build_full_trainset())
    predictions = svd.test(test)

    moviesList = []
    for m_id in ratings['movieId'].unique():
        moviesList.append((m_id, svd.predict(1000, m_id)[3]))

    # This portion takes the list of predictions and orders them in order
    # of most likely to be liked by the user to least.
    ranked_movies = sorted(moviesList, key=lambda x:x[1], reverse=True)

    # This takes in the list of predicted movies, the DataFrame of movies,
    # and a number reguarding how many movies to show (starting from the top)
    return recommended_movies(ranked_movies,movies,5)
```

In [15]:

```
#### Running this cell will start the questionair.
# It takes in the DataFrame of movies, a number for how many user inputs
# it requires, and the genre you wish to get recommendations from
movie_recommender(movies, 4, 'Comedy')
```

```
Thank you for participating! In order to obtain your recommendations, please rate 4 movies.
      movieId               title                   genres
8610   118270  Hellbenders (2012)  Comedy|Horror|Thriller
On a scale of 1 - 5, how would you rate this movie? press n if you have not seen this movie. Press enter t
o submit your answer:
5
      movieId                title                        genres
3508     4795  Father Goose (1964)  Adventure|Comedy|Romance|War
On a scale of 1 - 5, how would you rate this movie? press n if you have not seen this movie. Press enter t
o submit your answer:
2
      movieId                                  title            genres
3470     4734  Jay and Silent Bob Strike Back (2001)  Adventure|Comedy
On a scale of 1 - 5, how would you rate this movie? press n if you have not seen this movie. Press enter t
o submit your answer:
3
      movieId                                    title  \
3363     4571  Bill & Ted's Excellent Adventure (1989)

                     genres
3363  Adventure|Comedy|Sci-Fi
On a scale of 1 - 5, how would you rate this movie? press n if you have not seen this movie. Press enter t
o submit your answer:
1
Recommendation #  1 :  277    Shawshank Redemption, The (1994)
Name: title, dtype: object

Recommendation #  2 :  602    Dr. Strangelove or: How I Learned to Stop Worr...
Name: title, dtype: object

Recommendation #  3 :  659    Godfather, The (1972)
Name: title, dtype: object

Recommendation #  4 :  906    Lawrence of Arabia (1962)
Name: title, dtype: object

Recommendation #  5 :  680    Philadelphia Story, The (1940)
Name: title, dtype: object
```

The use cases for this are ultimately pretty obvious. Movie streaming services are all but uncommon. Movie streaming services like Hulu and Netflix require user input to be able to recommend movies to users effectively. Even if it's as simple as recommendations base on movies similar to the most recent.

This system could work in tandem, using a 'likes' system to recommend movies upfront could help jumpstart streaming services give something accurate to begin with.

In [ ]:

```
```