MOVIE RECOMMENDATION SYSTEM

Robert Johnson

Three Phases of System

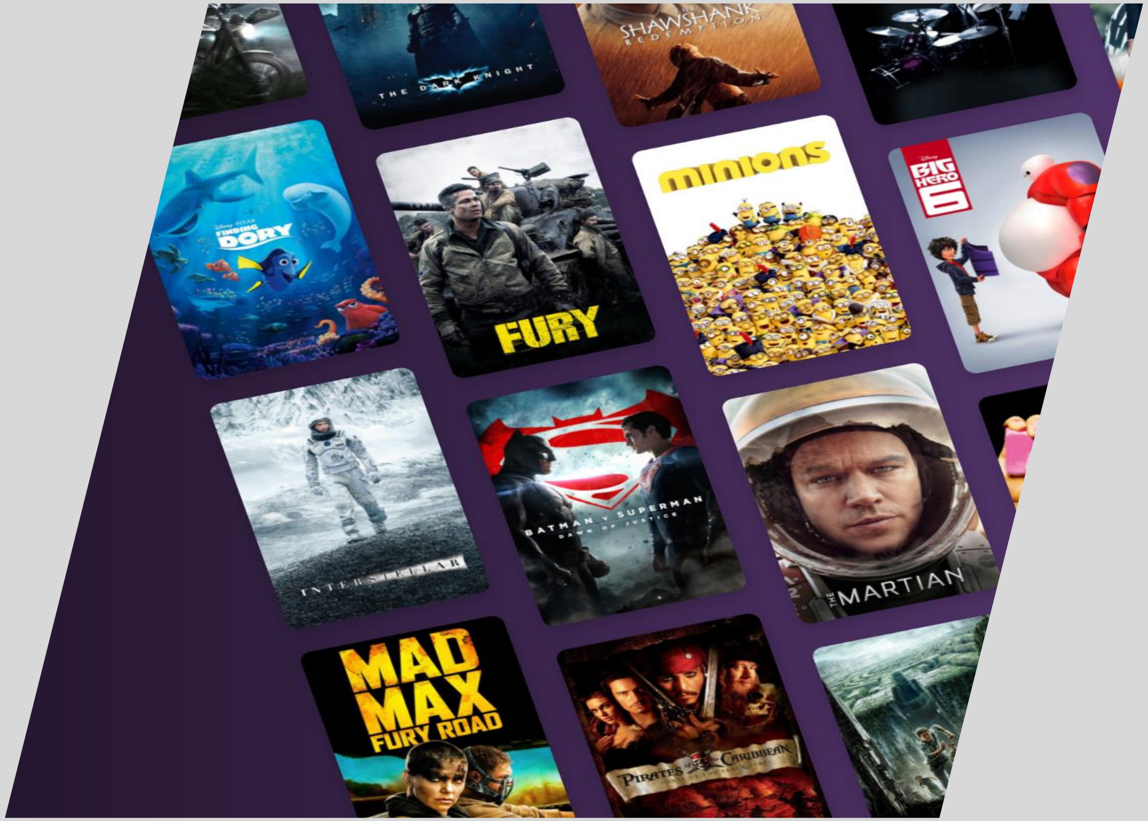
THE PROBLEM

THE PROCESS

THE RESULTS

THE PROBLEM ?

Build a movie recommendation system to help recommend movies on interests to the customers based on how they have rated the movies they have watched in our data set.



THE PROBLEM

For this project, I wanted to find a data set which provided us with plenty of observations and multiple variables. After a lot of research, we came across a data set from MovieLens. This data set fit the criteria our criteria of providing us with 100,000+ observations, and 6 variables. This data set was collected over a period of time, you can notice the movies that are rated in this data set range from the mid 1900's to early 2000's, providing us with a variety of movies.

THE PROCESS

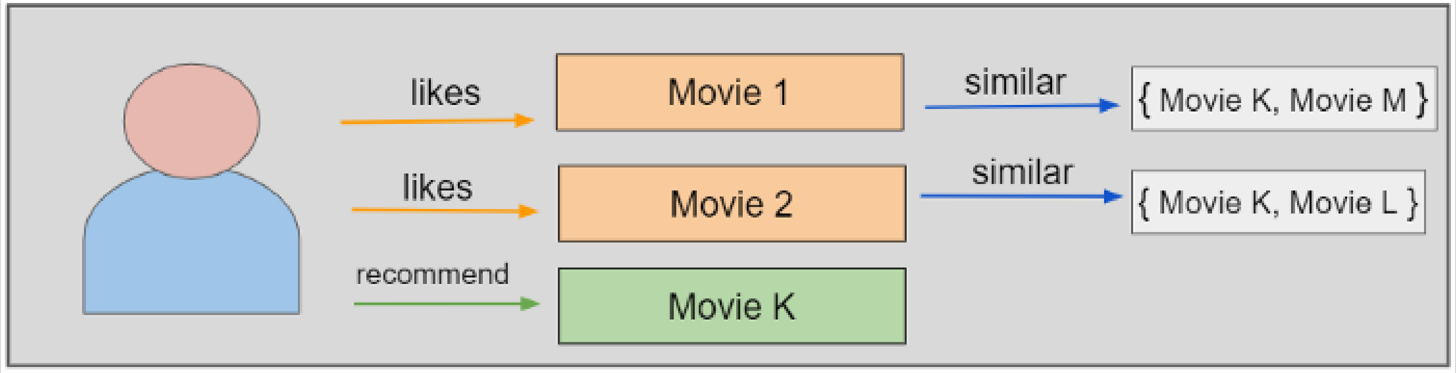
Singular Value Decomposition

- The format of the ratings matrix is one row per user and one column per movie using then converted that to a numpy array resulting in a sparsity level of 92.2%
- I used the Scipy SVDS function because it let's me choose how many latent factors.
- I multiplied U, Sum, and V^T back to get the rank k = 100.

movieId	1	2	3	5	6	7	10	11	14	16	...	76251	77561	78499	79132	80463	81591	81845	89745	91529	99114
0	6.559302	1.757697	0.482170	0.872076	0.509830	0.697081	2.976020	1.042342	0.499399	1.454533	...	1.278239	0.922927	0.756041	0.466786	0.993205	0.815785	0.760121	1.290980	1.478038	0.949495
1	0.151014	0.254994	0.480718	0.302252	0.176299	-0.202774	-0.742226	-0.726247	0.369260	0.281866	...	0.126826	0.453743	0.059608	-0.051221	0.396083	0.301666	0.043455	-0.435077	-0.314200	-0.307537
2	0.932169	0.196502	-0.038445	0.031548	-0.276534	0.505386	-0.126713	-0.059817	0.340507	0.160002	...	0.396815	0.245714	1.163325	2.799535	0.964277	1.397603	0.722184	0.967533	0.760296	0.765640
3	0.212598	0.725807	0.280377	0.589907	0.749680	0.294873	1.463197	1.197808	0.224493	1.140521	...	0.358639	0.312670	-0.016246	0.592838	0.087147	0.196339	0.147270	0.098349	0.067237	0.092485
4	0.213226	0.252179	0.321504	0.590974	-0.612023	0.575544	-0.021675	0.087825	0.144532	-0.129406	...	0.244307	0.078250	0.254014	-0.027543	0.257231	0.040386	-0.035558	0.058068	0.028157	0.129463

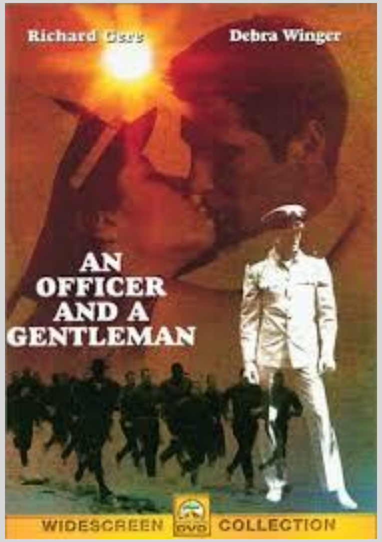
THE PROCESS

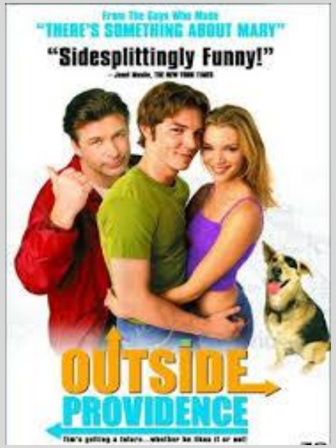
- I wrote a function to return the movies with the highest predicted rating that the specified user hasn't already rated.
- Here is the flow of the function get and sort the predictions using the SVD constructed matrix. Then get the user data via UserId for which we want to predict the top rated movies, merged with the Matrix for movie data, the final return will be the top movies recommendations that have not yet been watched.

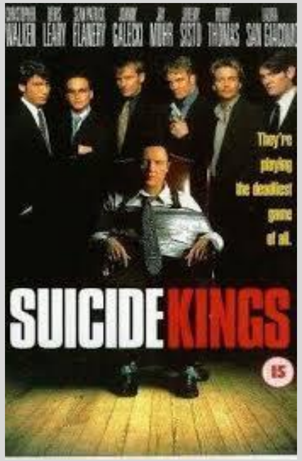


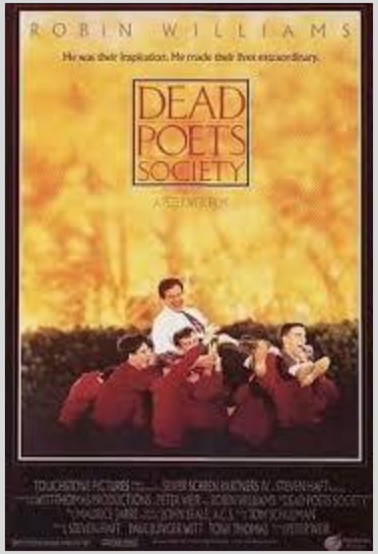
THE RESULTS

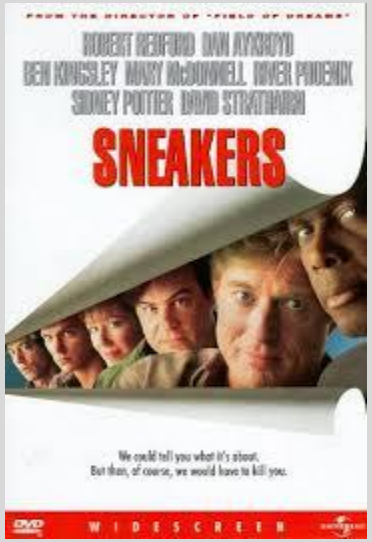
We first looked at User 810 top 10 rated movies





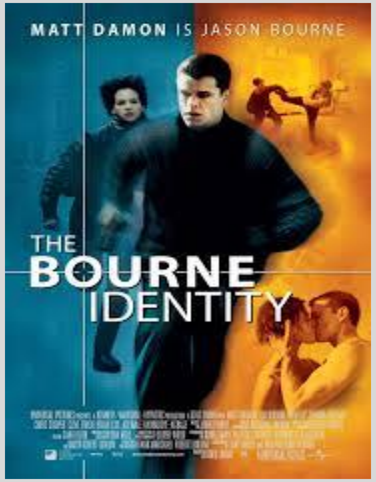


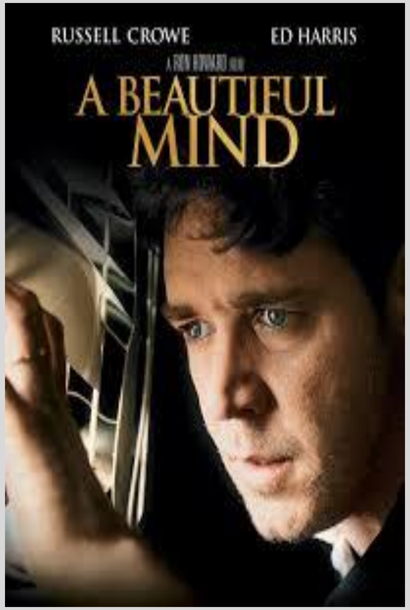




THE RESULTS

Here are User 810 top 10 recommendations movies





THE RESULTS

I used a similarity matrix, a pairwise distance calculation using the correlation computation do determine the predicted rating for the training and test set. Then compare that predicted number to the actual number then measure the Root Mean Square Error.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(Predicted_i - Actual_i)^2}{N}}$$

- User-based CF RMSE: 2.7501251498817356
- Item-based CF RMSE: 2.990641478787692

THE RESULTS

Conclusion

- Overall, this recommendation system works. The results according to the error check function have a relatively high root mean square error. However, the system seems to be effective.

Potential Future Work

- The process of creating a recommendation system was success this data was clean, so the process was not hard. There is still room to evaluate what movies someone would like even if they have never rated any movies. I think that recommendation system to would be the most useful for service providers.