Creating a Movie Recommendation Engine

An Examination of Collaborative Filtering Approaches

Final Project

BUAN 5310

Andrew Snodgrass

Mansi Agarwal, 2nd Team Member, 3rd Team Member

June 12, 2018

# INTRODUCTION

With the meteoric rise of streaming media services such as Netflix, Hulu, Pandora, and Spotify and the ever-expanding selection of content available to users on the internet, businesses face a new challenge. How can we personalize the user experience without engaging each and every customer? To a company like Netflix, whose 125 million subscribers stream more than 140 million hours of content every day, the answer is almost certainly in the data.

In this paper, we seek to examine the statistical methods that are used by companies today to automatically tailor the user experience and provide content recommendations. We begin by discussing the business problem and its relevance to the entertainment industry and beyond. Next, we examine some of the most popular statistical learning methods used in recommendation engines today. Finally, we conduct our own analysis using the MovieLens 100K data set and compare the results of our models.

# BUSINESS PROBLEM

For many web-based businesses, an effective recommendation engine is the key to success. With the vast amount of information and choices available to people on the internet, some method of filtering is required to avoid overwhelming users. This applies not only to streaming media providers, but also to e-commerce sites, advertising agencies, search engines, and news outlets. For this reason, companies invest a huge amount of resources into maintaining and improving their existing recommendation algorithms.

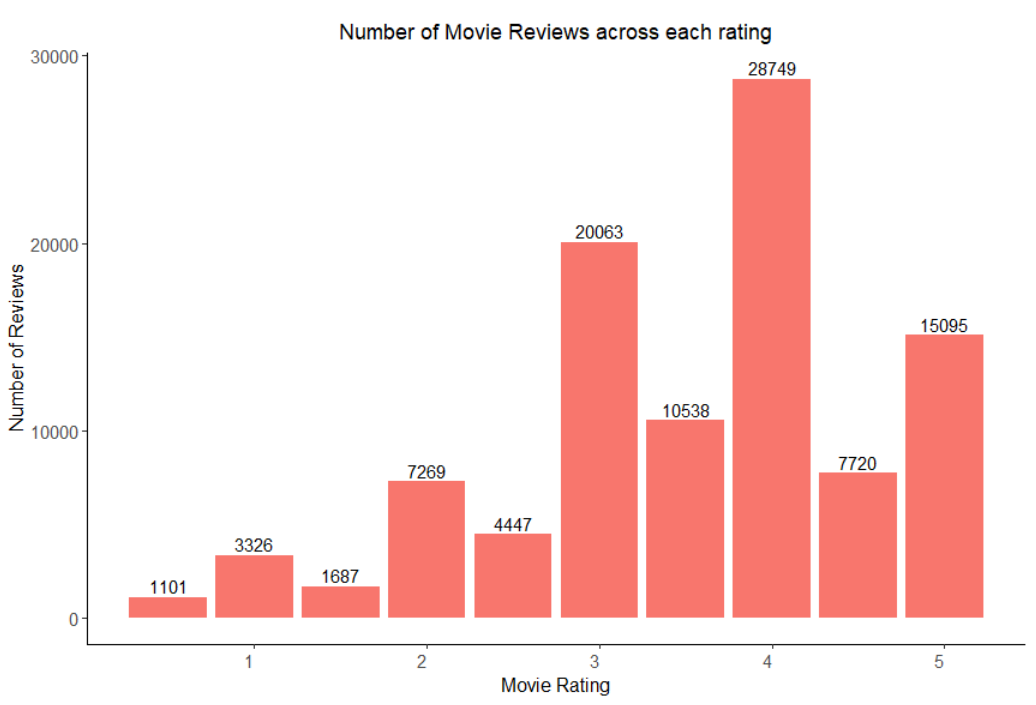
In 2006, Netflix famously released a data set containing roughly 100 million user movie ratings, offering a $1 million prize to whoever could best improve upon Netflix’s own Cinematch™ recommendation engine. This high-profile endeavor to crowdsource new methods indicated to the world that companies were eager to innovate upon existing algorithms. With this in mind, we sought to better understand the methods that companies are currently using to provide recommendations to their users. Thus, our business problem is as follows: given a data set depicting information about movies and their ratings from various users, how can we best recommend content to each user? We address this problem by attempting to predict the rating that each user would give to a movie and benchmark our success by measuring the root-mean-square error (RMSE) between our predictions and actual ratings.

# THE DATA

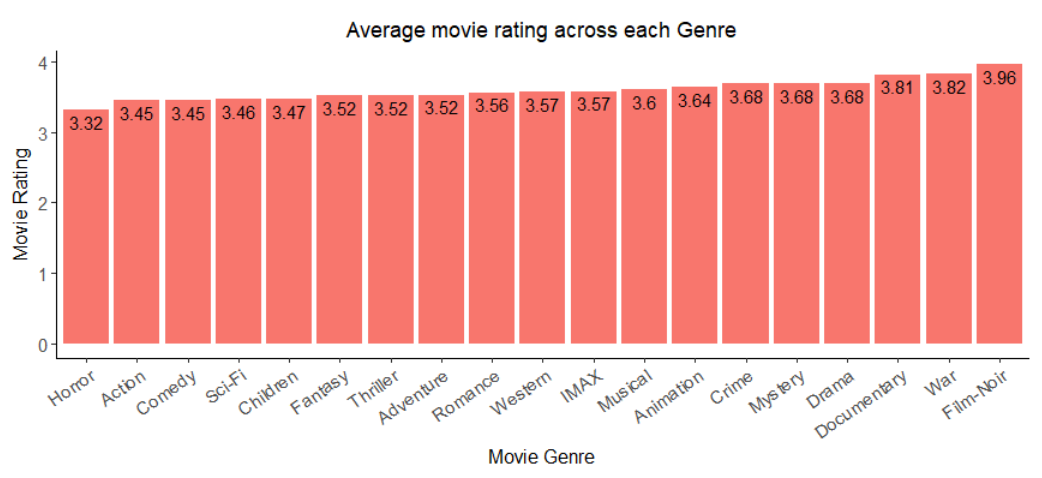
For our analysis, we use the 2016 MovieLens 100K data set. MovieLens is a web-based movie recommendation service operated by GroupLens, a research lab at the University of Minnesota. It is a non-commercial service which uses the data generated by its users to advance social computing research. Our data set contains 100,004 ratings for 9,115 movies, as well as information about genres and user-input text tags. Ratings were generated by 671 users, each of whom has provided at least 20 ratings between January 1902 to October 2016. These users represent a randomly selected subset of the full data, which contains 26 million ratings, 45,000 movies, and 270,000 users. The data is contained in 3 tables: ratings, movie information, and user tags.

# EXPLORATORY DATA ANALYSIS

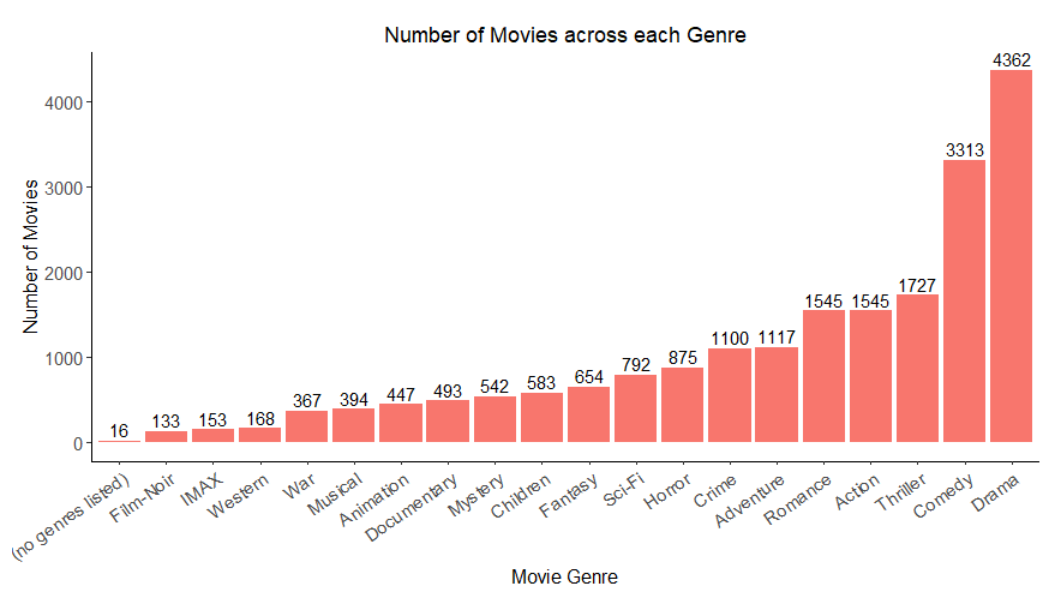
To gain a better understanding of the data, we begin by conducting an exploratory data analysis (EDA). Looking at our user ratings, we observe that the average rating is 3.54. Given that we are working with a rating system ranging from 0.5 to 5 stars, this indicates that users are somewhat generous in their evaluations. Figure 1 below depicts the frequency of each rating for the full data set. Interestingly, although the average total rating is roughly 3.5 stars, the most frequent rating given by users is 4 stars. In fact, as we see here, users are far more likely to rate movies using full stars than half stars in general.



Diving a little deeper, we ask the question: how do average ratings differ across genres? To answer this, we join the data from our movie information table to our ratings table. In total, we find that the movies in our data set are labeled according to 19 different genres, with some movies carrying as many as 10 genre tags and some movies carrying none. Figure 2 below depicts average user ratings by genre. We observe that average ratings are between 3 and 4 stars for all genres. Interestingly, the top 4 genres by average rating appear to be somewhat niche interests, including one “genre” which simply serves as a filler tag for movies which have no genre. In contrast, some of the genres we would expect to be more popular such as action and comedy have lower average ratings. This suggests to us that the ratings of less popular genres likely benefit from a dedicated fan base.



To test this theory, we decide to examine the frequency with which each genre appears in the data set. We expect that some of the more niche genres such as Film-Noir, War, IMAX, and Western will have a lower representation in our data, while more broad genres such as Action, Comedy, and Drama will have a higher representation. Figure 3 below depicts the number of movies across each genre in our data.



The results of this graph appear to support our hypothesis. As we can see, no genre, Film-Noir, IMAX, Western, and War are the least-represented categories in our data, while drama, comedy, thriller, action, and romance are the most-represented categories.

# STATISTICAL METHODS

To address our business problem, we consider the various methods that are used in modern recommendation engines. A wide range of specialized algorithms exist, but in general, they can be boiled down into three main categories: non-personalized, content-based, and collaborative filtering.

Non-personalized recommendations are perhaps the simplest approach, in that they do not consider individual user tastes. Instead, they simply make the same recommendation to all of their users based on the universally most popular and highest rated items in their library. In the case of movie recommendations, we might achieve some moderate success with this approach, but it does not align with the end goal of providing a tailored experience to our customers.

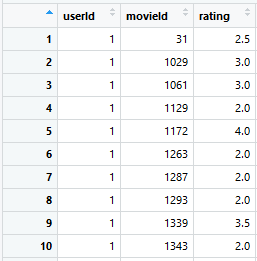
Content-based recommendations consider the attributes of each item and attempt to determine which attributes are favorable to each user based on their previous ratings. In our case, the attributes available for consideration are genre, year of release, and in some cases, user text tags. Thus, a content-based model might determine that Customer X tends to watch a lot of comedies and rates them highly, on average. Based on this information, it would be more likely to recommend comedies to Customer X in the future. Unfortunately, this method is much more successful in situations where items are paired with a wide variety of detailed tags. In our case, simply knowing that a movie is a Drama released in 1996 does not get us much closer to knowing if it will appeal to a particular user.

Collaborative filtering is a recommendation method that attempts to quantify the similarity between users and/or items in a system to provide recommendations. There are two main forms of collaborative filtering: user-based and item-based. In user-based collaborative filtering, items receive a rating for a particular user based on how other similar users have rated that item. Similar users are determined by previous interactions with items in the system. For instance, if Customer A and Customer B both liked Item C, and Customer A likes Item D, user-based collaborative filtering might recommend Item D to Customer B. Item-based collaborative filtering employs a similar approach, with a focus on identifying similar items rather than similar users. Given the nature of our data, we believe that a collaborative filtering approach is the most suitable method to begin with.

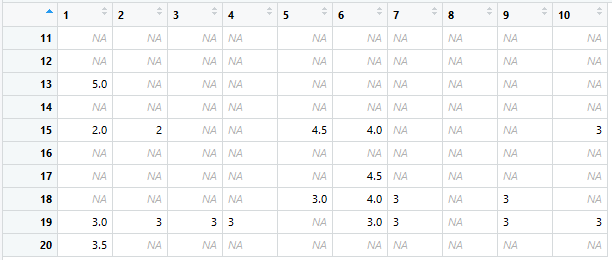
# BUILDING OUR MODELS

In order to employ a collaborative filtering algorithm, we must reshape the data in our ratings table. From the initial data set, in which each row represents a user-item rating pair, we spread the table into a sparse matrix. The resulting matrix contains 671 rows and 9,125 columns, in which each row represents a user and each column represents a movie. The data contained in each cell represents the rating that each user has given to each movie. The figures below illustrate an example of the initial data set and the resulting sparse matrix.

Initial data set:

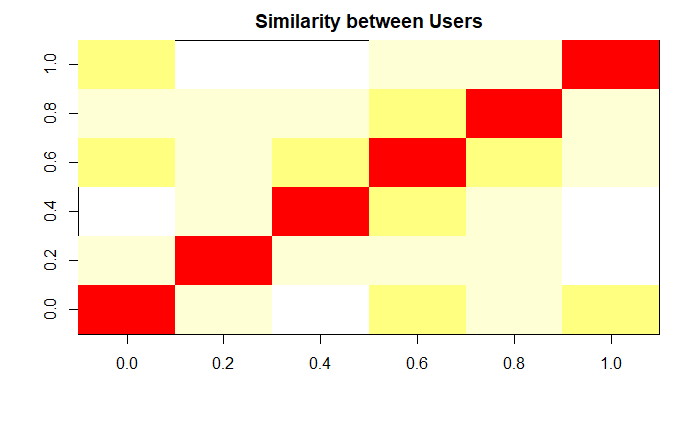


Sparse matrix:

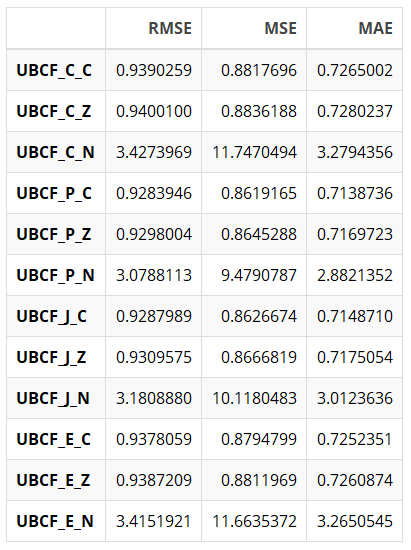


The example above leaves little question as to why we call this a “sparse” matrix. Cells only contain values for which a user has rated any given movie. As even the most dedicated users will not have come close to watching and rating all 10,000 movies in our data set, we end up with many empty cells. This is not a problem for our algorithms, however. In fact, the empty spaces themselves provide some valuable information. If we consider each user as a vector of the ratings they’ve given (or not given), one measure of similarity between users is simply whether they’ve viewed and rated the same movies.

With our sparse matrix created, we are ready to determine the degrees of similarity between users. This is accomplished by calculating a “distance” factor between users, for which there are several different approaches. The most popular methods for measuring distance are: Euclidian distance, Pearson coefficient, Jaccard coefficient, and Cosine similarity. During our analysis we experiment with each of these methods, but for the sake of brevity, we will not attempt to define them herein. We have included an article in our references entitled “Mining Similarity” which explains each of these methods in greater detail. The figure below visualizes the outcome of one such method, Cosine similarity, when comparing a sample of 6 users to each other. The matrix can be interpreted in a similar fashion to a correlation matrix, in which a red square indicates a perfect correlation between users, a white square indicates no correlation, and darker shades of yellow indicate greater levels of correlation compared to lighter shades of yellow. As with a correlation matrix, the diagonal line of red squares represents each user being compared to themselves, resulting in perfect correlation.



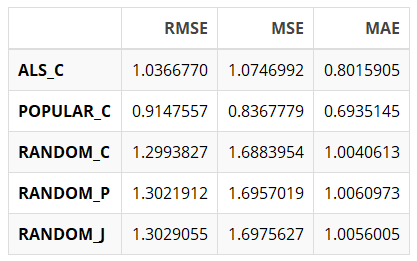
To create our recommendation models in R, we utilize the “recommenderlab” package. This package allows us to train collaborative filtering models on our data using different algorithms, similarity metrics, and normalization approaches. For our first set of models, we use a 70/30 split to create training and test sets. Later, we experiment with k-fold cross validation to determine if we can improve the RMSE of our models. We begin with a user-based collaborative filtering (UBCF) approach, experimenting with each of the similarity metrics outlined above and each of 3 normalization methods (central, z-score, and non-normalized). By considering each of these combinations, we end up with 12 different UBCF models. The table below compared the error rate for each of these models.



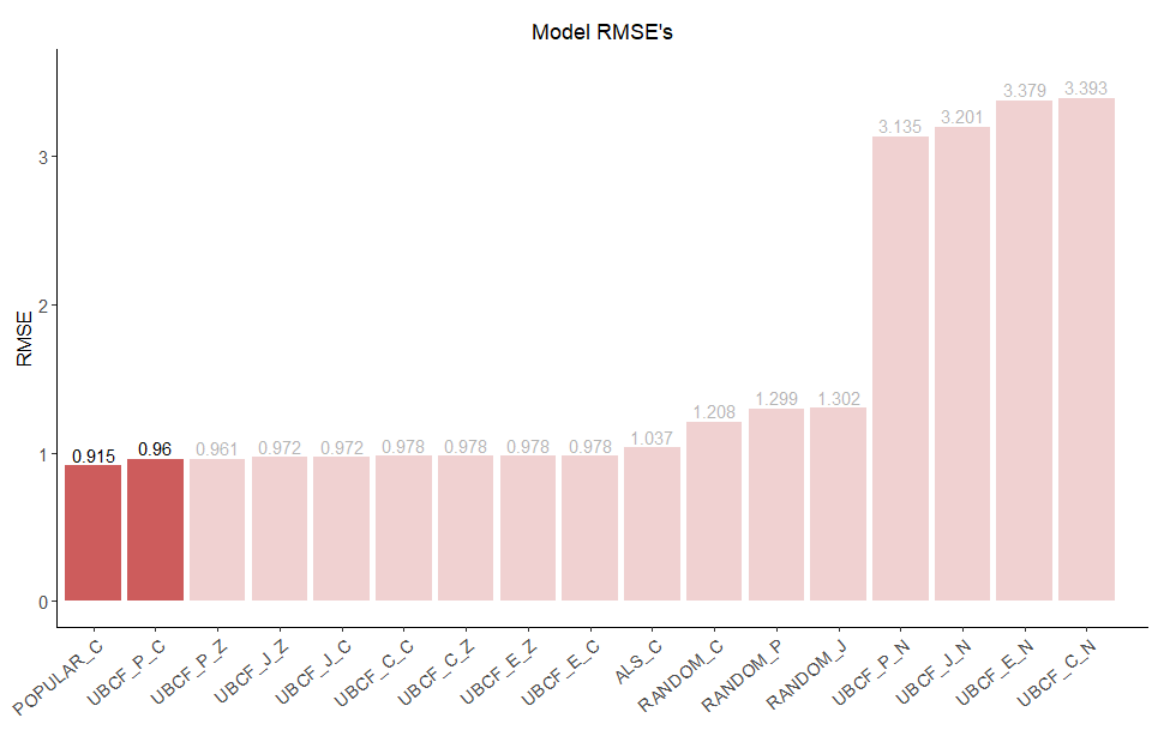
In this table, our models are labeled as follows: Method\_Similarity \_Normalization. Looking at our RMSEs, the first thing we notice is that our non-normalized models (indicated by an N) have significantly higher RMSEs than those with central or z-score normalization. Our normalized models demonstrate somewhat similar RMSEs to each other, but it appears that the models using Pearson coefficient perform marginally better than the others.

With a set of baseline models created and a top-performing RMSE of roughly 0.96, we now have a benchmark for our new models. Our next logical step is to create an item-based collaborative filtering (IBCF) model for comparison. Unfortunately, due to the large number of items in our data set, our attempts to create an IBCF model resulted in unreasonably long processing times and we were unable to successfully train one. As a result, we conclude that IBCF may not be a suitable approach for this data set.

Next, we train a set of models using three new methods: alternating least squares (ALS), Popular, and Random. ALS is a popular collaborative filtering method which attempts to deal with null matrix values by considering latent (unobserved) factors in the data. Popular is a method which adjusts the weight of its ratings according to item popularity, where popularity is defined as number of users who have interacted with (rated) the item. Finally, the random method simply produces random recommendations for items. The error rates for these new models are depicted in the table below.



As we might expect, the random recommendation method performed poorly. We were also somewhat surprised that ALS did not achieve a competitive RMSE. However, the popular method provided a somewhat significant improvement over our previous benchmark, with an RMSE of roughly 0.915. As such, the popular method is currently our best performing model. Our model RMSEs thus far are illustrated in the table below.

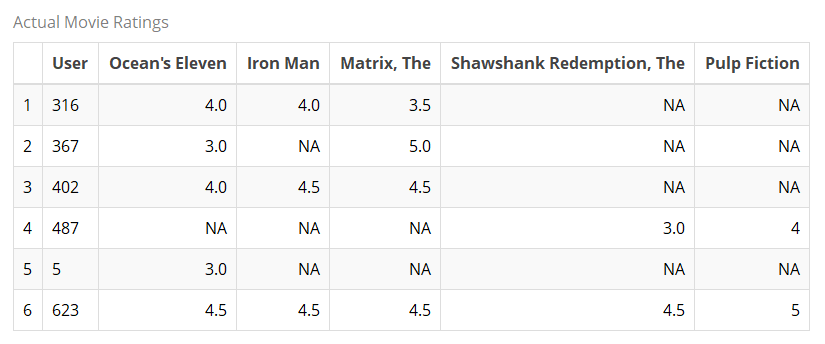


# Results

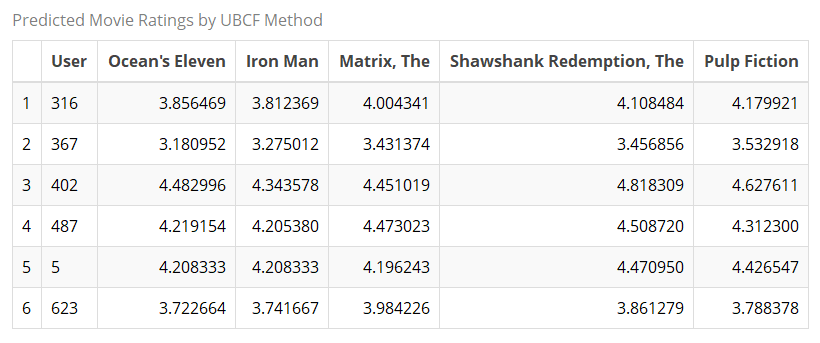
## **Predicted Ratings**

At this point, we have trained a variety of models utilizing different approaches. To better illustrate the predictions generated by these models on which our RMSEs are based, we provide a few examples comparing actual user ratings for some of our favorite movies to ratings predicted by a few of our models.

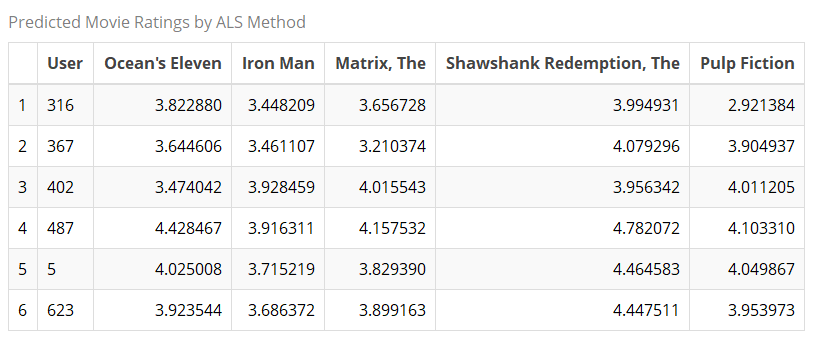
Actual ratings:



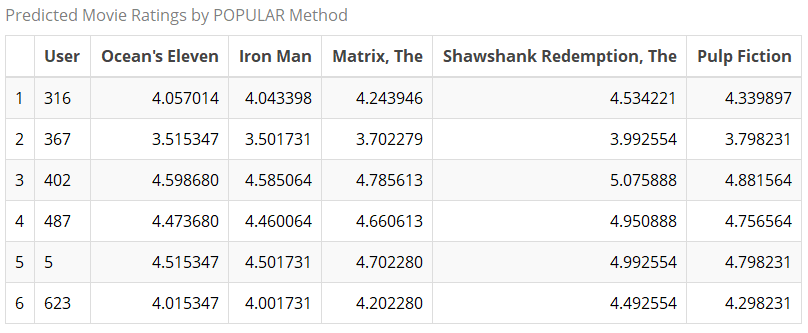
UBCF, Cosine, Center normalized:



ALS, Cosine:

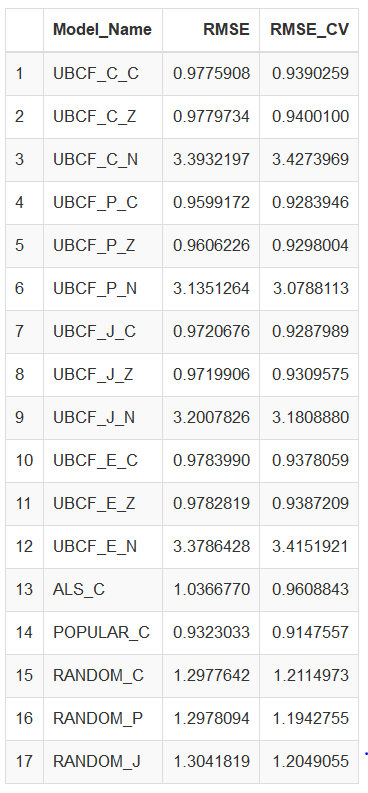


Popular:



As we can see, predicted ratings are between 3 and 4 stars in most cases. There does not appear to be a consistent trend of UBCF or ALS overestimating or underestimating ratings. However, we do observe an interesting phenomenon among the popular ratings. While the movies depicted in these tables are some of our own favorites, it’s also true that they are some of the most popular movies of all time. As a result, the popular method provides much higher ratings to these movies than the UBCF or ALS methods.

Next, we attempt to optimize our models using k-fold cross validation. After some experimentation, we determine that setting k = 12 provides the best results, on average. The tables below demonstrate the change in RMSE between the training/test set approach and k-fold cross validation with k = 12 for each of our models.

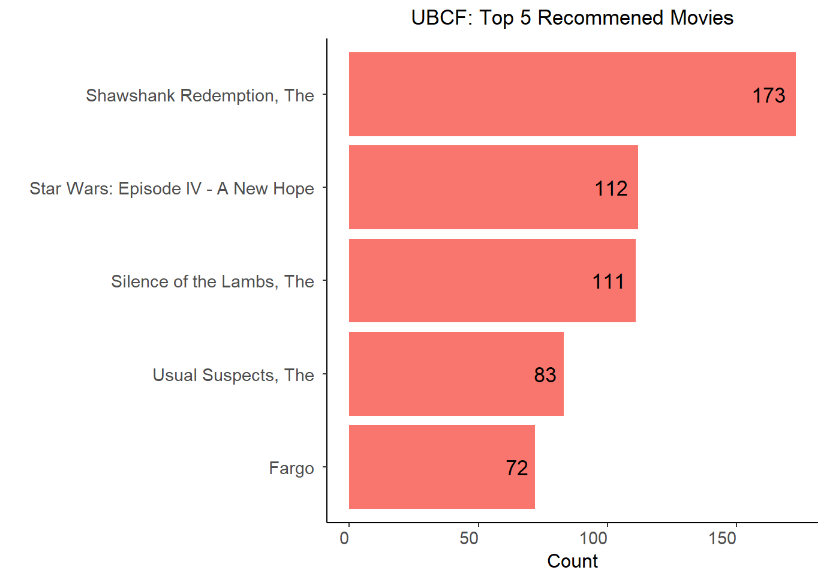


We observe that in most cases, employing k-fold cross validation improves our RMSE substantially. Note that the RMSE for the popular model does not change at all. Based on our new results, it appears that our UBCF models are now somewhat competitive with the popular model. In addition, cross-validation has substantially improved the RMSE of the ALS model. However, the popular model still achieves the lowest RMSE.

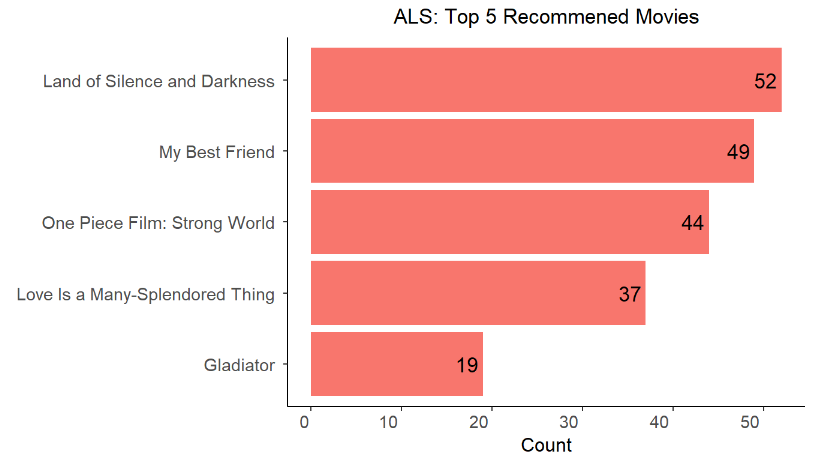
## **Recommended Movies**

We also used our trained models: UBCF, ALS and POPULAR (all with Cosine Distance and Central Normalized) to recommend top 6 actual movies for each user. Our initial hypothesis here is that the recommended movies for a user should be same as the top rated (predicted) movies for that user.

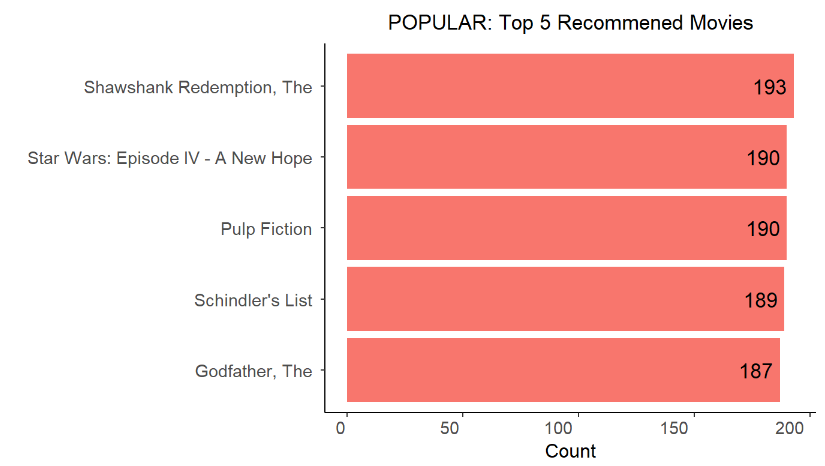
We find that with each model, there is a different set of movies that are most frequently recommended. UBCF recommends “The Shawshank Redemption” to 173 users and is the most frequently recommended movie followed by “Star Wars: Episode IV – A New Hope” (to 112 users) and The Silence of the Lambs (to 111 users).



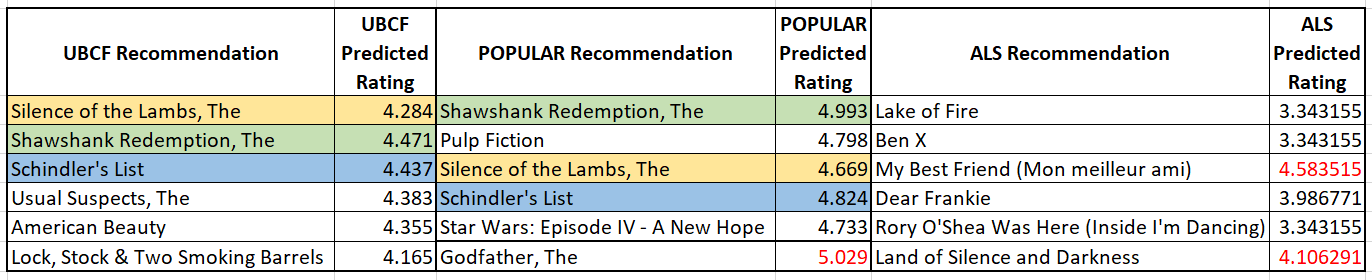
On the other hand, ALS recommends “Land of Silence and Darkness” most frequently (to 52 users) followed by “My Best Friend” (to 49 users) and “One Piece Film: Strong World” (to 44 users).



Results from POPULAR are somewhat close to UBCF, where in “The Shawshank Redemption” (to 193 users) is most frequently recommended movie followed by “Star Wars: Episode IV – A New Hope” (to 190 users) and Pulp Fiction (to 190 users).



We also compared the movie recommended to a particular user (User 5) from these various models in the table below. We find that movies recommended by POPULAR are mostly watched and rated by users and have overlaps with UBCF recommendations. ALS recommendations are hard to make sense of which is in line with the RMSE for this model.



We also find that the movie recommendations are not arranged in the order of predicted ratings i.e. we have movies that are recommended as second choice but have higher predicted rating than the first choice. This negates our initial belief that that the recommended movies for a user should be same as the top rated (predicted rating) movies for that user.

# CONCLUSION & RECOMMENDATION

Based on our results, we are inclined to suggest that the popular model will perform best given the data available. However, we believe that the UBCF and ALS models would be substantially improved with a larger data set. With this in mind, we suggest proceeding with the implementation of the popular model and conducting further analysis of both the ALS model and the UBCF model with Pearson coefficient on larger variations of the data set. Given the opportunity to draw similarities among a larger user base, we believe that these models offer the potential for more scalability and a more personalized user experience than the popular model.

**REFERENCES**

Dataset: <https://grouplens.org/datasets/movielens/latest/>

Grover, Prince. “Various Implementations of Collaborative Filtering.” *Towards Data Science*, 28 Dec.2017, [towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0.](https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0)

Hahsler, Michael. “Recommenderlab: A Framework for Developing and Testing Recommendation Algorithms.” *CRAN*, [cran.rproject.org/web/packages/recommenderlab/vignettes/recommenderlab.pdf.](https://cran.r-project.org/web/packages/recommenderlab/vignettes/recommenderlab.pdf)

Herlocker, Jonathon L, et al. “Explaining Collaborative Filtering Recommendations.” *GroupLens*, [grouplens.org/site-content/uploads/explain-CSCW-20001.pdf.](https://grouplens.org/site-content/uploads/explain-CSCW-20001.pdf)

Isinkaye, F.O, et al. “Recommendation Systems: Principles, Methods and Evaluation.” *Egyptian Informatics Journal*, vol. 16, no. 3, Nov. 2015, pp. 261–273. *ScienceDirect*, <https://www.sciencedirect.com/science/article/pii/S1110866515000341>

Mason, Moya K. *Short History of Collaborative Filtering*. <http://www.moyak.com/papers/collaborative-filtering.html>

*Mining Similarity Using Euclidean Distance, Pearson Correlation, and Filtering*. [mines.humanoriented.com/classes/2010/fall/csci568/portfolio\_exports/mvoget/similarity/similarity.html.](http://mines.humanoriented.com/classes/2010/fall/csci568/portfolio_exports/mvoget/similarity/similarity.html)

Verreet, Bregt. “The Alternating Least Squares Algorithm in Recommenderlab.” *Infofarm*, <https://www.infofarm.be/articles/alternating-least-squares-algorithm-recommenderlab>