Movielens Report

Ariel Cooper

2024-06-06

MovieLens Capstone

Background

When streaming shows online, there will be the ubiquitous recommendations for what to watch next. What you are show is customized to you, enticing you to watch more content tailored to your preferences and past behavior. Along with this the streaming service benefits by getting you to watch more of their content. In 2006, Netflix released a challenge to beat their current algorithms by at least 10%. To the winner would go a prize of $1,000,000.

To create the recommendations, challengers were given a training set of data of over 100,000,000 rows. Each row included information such as, but not limited to, a user ID, movie title, and a rating from 1-5. Netflix would have a second set of samples of around 2.8 million, which the contestants would need to predict the rating using the training set they were provided. The measure of success would be to get the lowest root mean squared error(RMSE), which will be explained further in the next section. To beat Netflix's algorithm, a RMSE of 0.8563 or lower would need to be achieved. (T¨Oscher, A., & Jahrer, M., 2009)

RMSE what is it and how it is calculated? When trying to calculate how well an algorithm is performing we need some kind of metric to base it off of. Is just guessing randomly better than your carefully crafted code? There are many ways to create this calculation and it all depends on what you are trying to measure and the data it comes from. For this project we will be using RMSE.

This formula is calculating the average distance from predicted values to the actual value. For the most accurate prediction, we want the RMSE to be as low as possible since it is indicating our predicted values are not far from the actual. (Bobbitt, 2021)

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

N is the sample size

u, i refers to the index, so in this case it the review for user u and movie i

$\hat{y}_{u,i}$ is the predicted rating for user u and movie i

$y_{u,i}$ is the true rating for user u and movie i

$\sum_{u,i}$ is the sum of all results over all users and movies

Exploring the Data

We will be using a subset of the MovieLens data with 1M entries due to time and hardware constraints. 10% of the data will be separated out as our test for accuracy. We cannot use this set of data during creation of our algorithm, similar how we would not know exactly how someone would rate a movie in real life. But we could try to figure it out based on past behavior of the person and others. This mystery subset will only be used to check our results at the end of the testing.

Overall the downloaded movielens data has already been cleaned from any entries contain NULL values. It contains several columns (Table 1).

Table 1: Original columns provided

Column	Data Type	Description
userId	integer	unique identifier for a person providing ratings
movieId	integer	unique identifier for each movie
title	chr string	name of the movie
rating	number	between 0-5 in 0.5 increments representing the rating the user has provided. 5 being the highest
genres	chr string	genres of the movie, separated by a '|' character
timestamp	integer	UNIX timestamp of when the rating was created

Two additional columns were created from the existing data to be used later (Table 2).

Table 2: Created columns

Column	Data Type	Description
release_year	integer	year movie was released, extracted from title string
years_since	integer	year the rating was created extracted from timestamp - release year

In the edx table, there are a total of 9,000,055 ratings, from 69,878 users for 10,677 movies. Based on those numbers, not every user is rating every movie. It is only about 1.2% of the total possible ratings, but we would not expect every person to have watched and rated every movie.

```
# Number of movies
n_distinct(edx$movieId)
# Number of users
n_distinct(edx$userId)
# Total ratings
nrow(edx)
```

Given that not every user is rating everything, the distribution of how frequently a user rates a movie varies highly (Figure 1). Only a small portion of users have more than 100 ratings recorded. Then looking at the spread of the ratings based on the user (Figure 2), it shows that most users will rate a movie between 3-4 stars. This shows a bias that an individual creates and should be accounted for in the algorithm.

Figure 1: Frequency of users with a given number of ratings

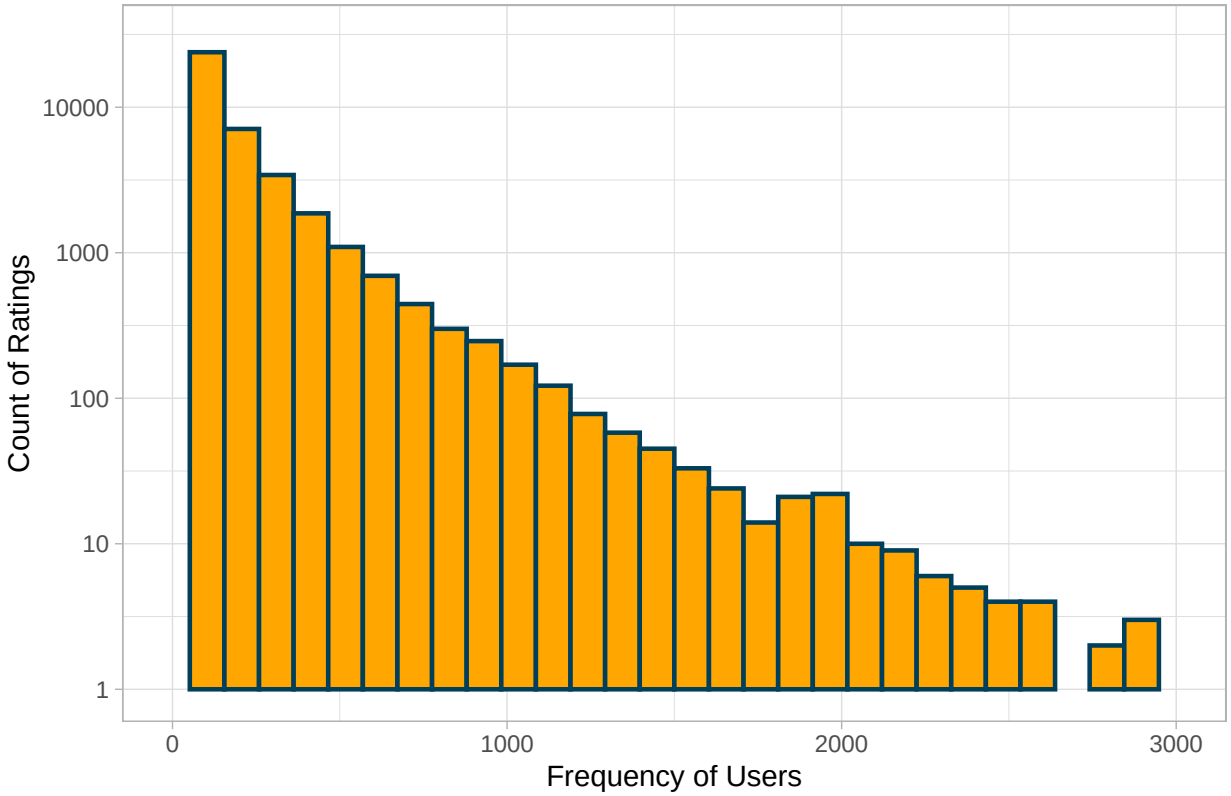
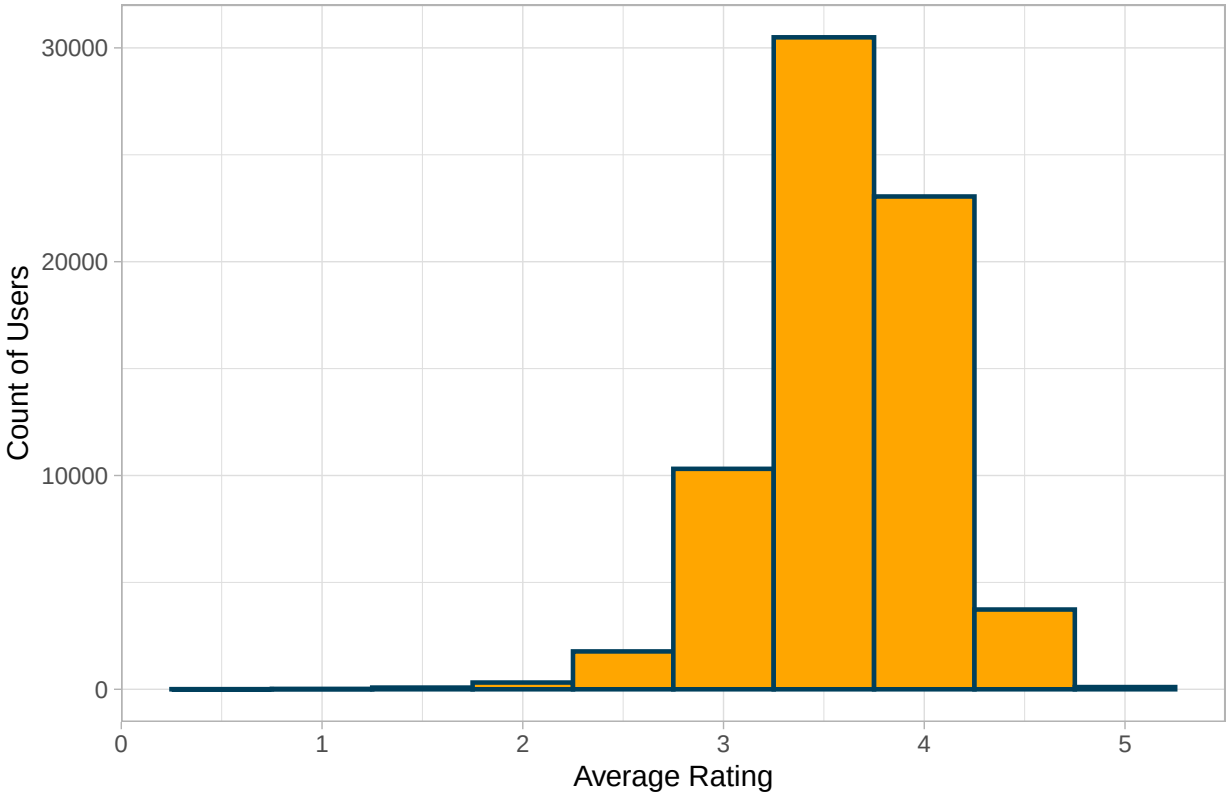


Figure 2: Frequency of the average rating based on users



Similar to the bias created by a particular user, we can see that a particular movie will have its own trend of most ratings being between 2.5-4 stars(Figure 3). Also we can see that more popular movies(i.e. more ratings) tend to have a higher ratings(Figure 4). This effect should be accounted for when predicting the ratings.

Figure 3: Frequency of the average rating of a movie

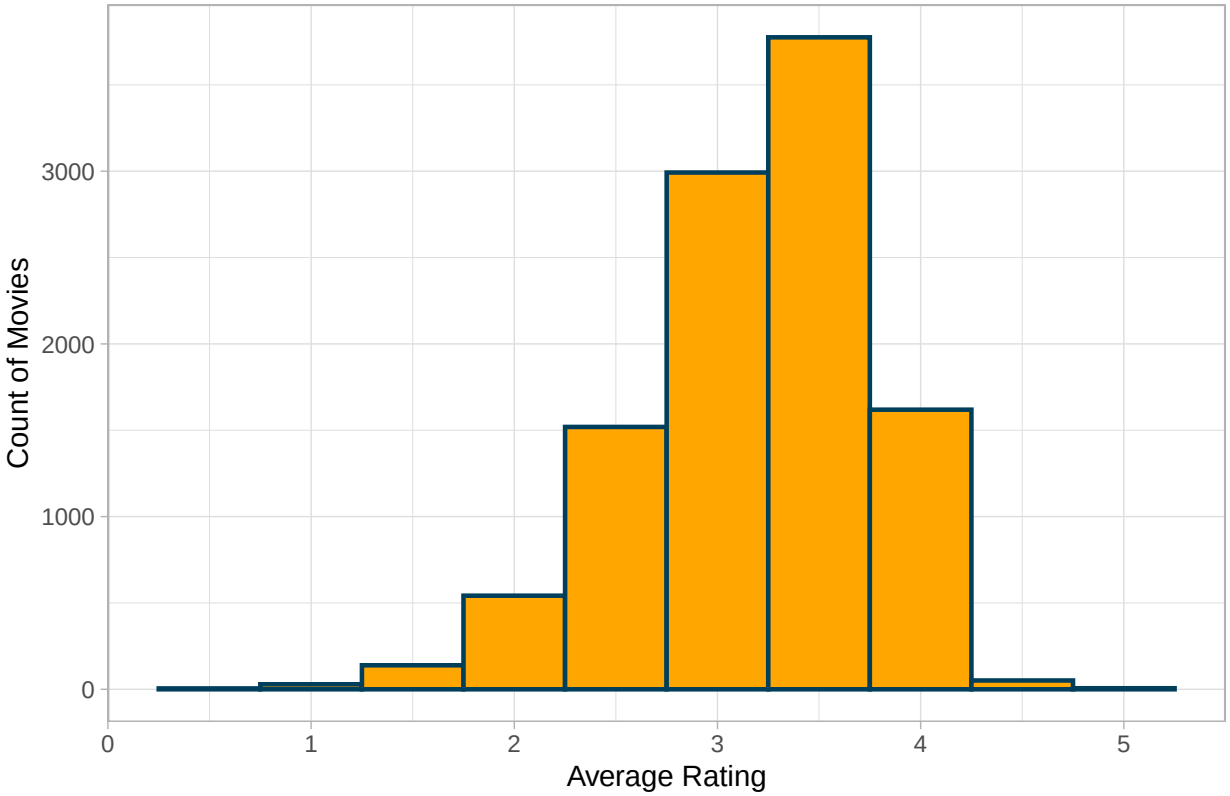
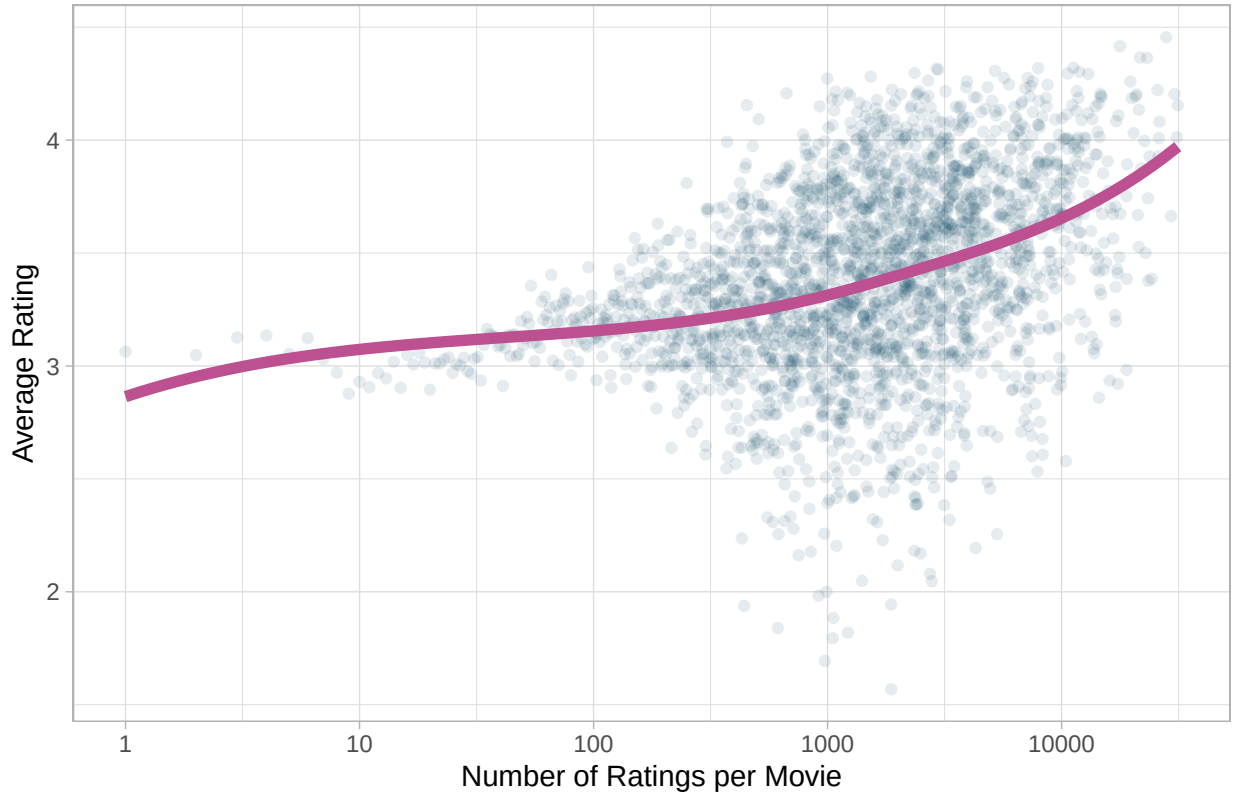
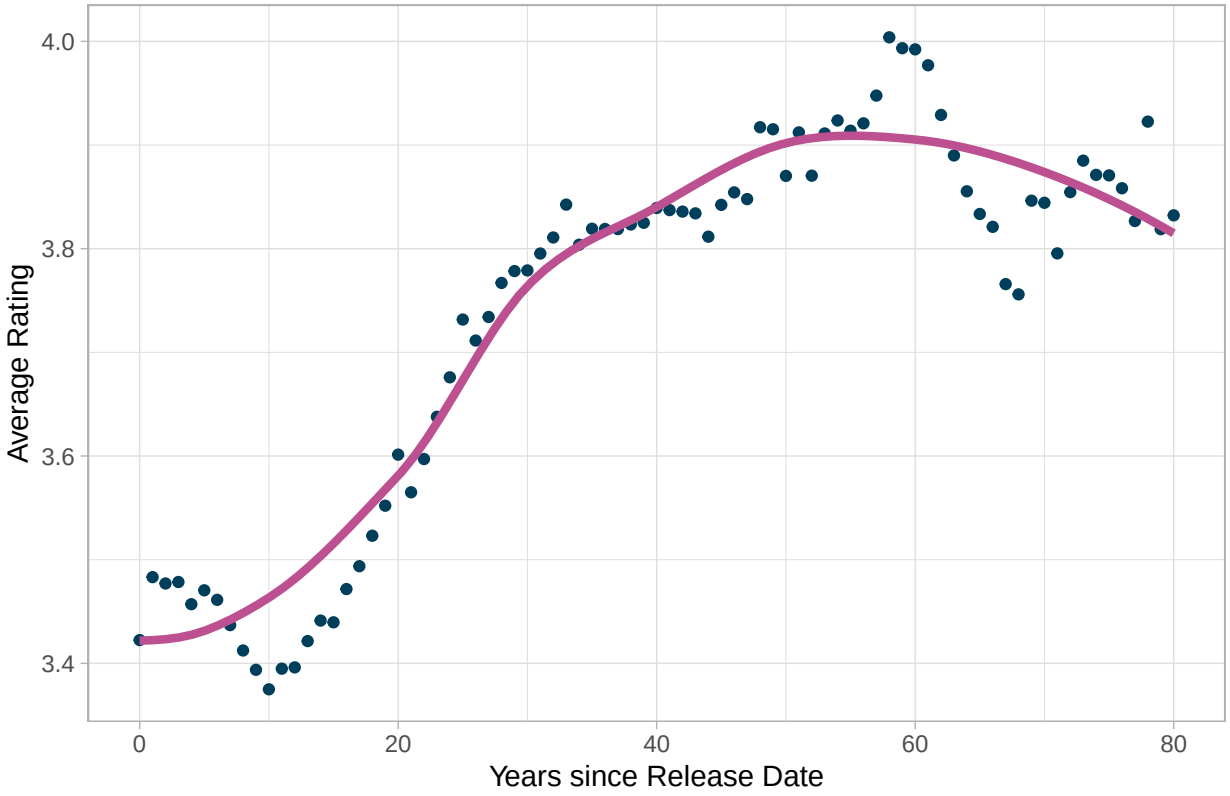


Figure 4: Average rating based on the frequency a movie is rated



After calculating how long ago the rating is from the release year, a trend is revealed that there is higher ratings as time goes on (Figure 5). Time ranges with less than 1000 ratings were removed due to creating outliers.This trend may be due to if a movie is still getting frequent ratings after the first 5 years, then it could be called a classic and is more highly rated.

Figure 5: Comparison of time since release vs the average rating



Pre-processing

Before splitting up the data between the final holdout test set, two columns were created (Figure 2). The release year and the number of years between the release and the rating.

With the available data it is split up into two again. A training set to give the algorithms something to work off of and a testing set to test RMSE against before applying to our original mystery set.

A list of indices was created for each userId that contained 90% of their ratings in one group and 10% in another. The 10% was used for the testing set, 90% for training. They were then compared to make sure both sets contained the same userId's and movieId's, as if any were missing from the training set it would create issues later on.

```
#creates sets of each user and then takes 20% of each user for testing
set.seed(2006)
indexes <- split(1:nrow(edx), edx$userId)
test_ind <- sapply(indexes, function(ind) sample(ind, ceiling(length(ind)*.2))) %>%
  unlist(use.names = TRUE) %>% sort()

training <- edx[-test_ind,]
temp <- edx[test_ind,]

# Make sure same movieId in testing and training
testing <- temp %>%
  semi_join(training, by = "movieId")
```

Model	rmse
Average Rating	1.06037

Model	rmse
Movie effect model	0.9441609

```
# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, testing)
training <- rbind(training, removed)
```

Methods tried

Guessing The most basic method used is to guess the same rating for all movies, regardless of any other factors. The best fit for this is to take an average of all ratings in the data. We'll call this variable μ. Putting this in a general formula, it comes out to our prediction $\hat{y}$ for any given rating equaling μ.

$\mu = 3.511909$

The formula being:

$\hat{y} = \mu$

Movie Bias

Now we know intuitively and from exploring the data before that not every movie will have the same rating. Some have better ratings on average than others and others less than the average. To account for this we will add a new variable b_i. This is the bias created by a particular movie i. To calculate this we'll see how far from μ a paritcular movie is on average, and then use that number when making a prediction for that movie. (Irizarry, n.d.)

$Y_i = \mu + b_i$

User Bias

Just as each movie will have its own effect on the rating, each user will have their own effect indicated by b_u. For simplicity we just add the two together in the current formula, but later on we will adjust for b_i when calculating b_u. By combining the two effects we significant drop in the RMSE, so both the movie and the user are confirmed to be able to be used as predictors.

$\hat{y}_{i,u} = \mu + b_i + b_u$

Reg Movie + User When looking at the top rated movies, the results seem odd, they are not well known titles(Table 3). Looking closer, they only have a single rating. Intuitively we would expect a truely popular movie to have more ratings, so to account for this we are going to adjust b_i and weight it based on the number of reviews n_i. Additionally, adjusting with a chosen value λ, the RMSE can be reduced. To find the ideal λ, a varitity of values will be tested to find the one that reduces the RMSE the most (Figure 6)

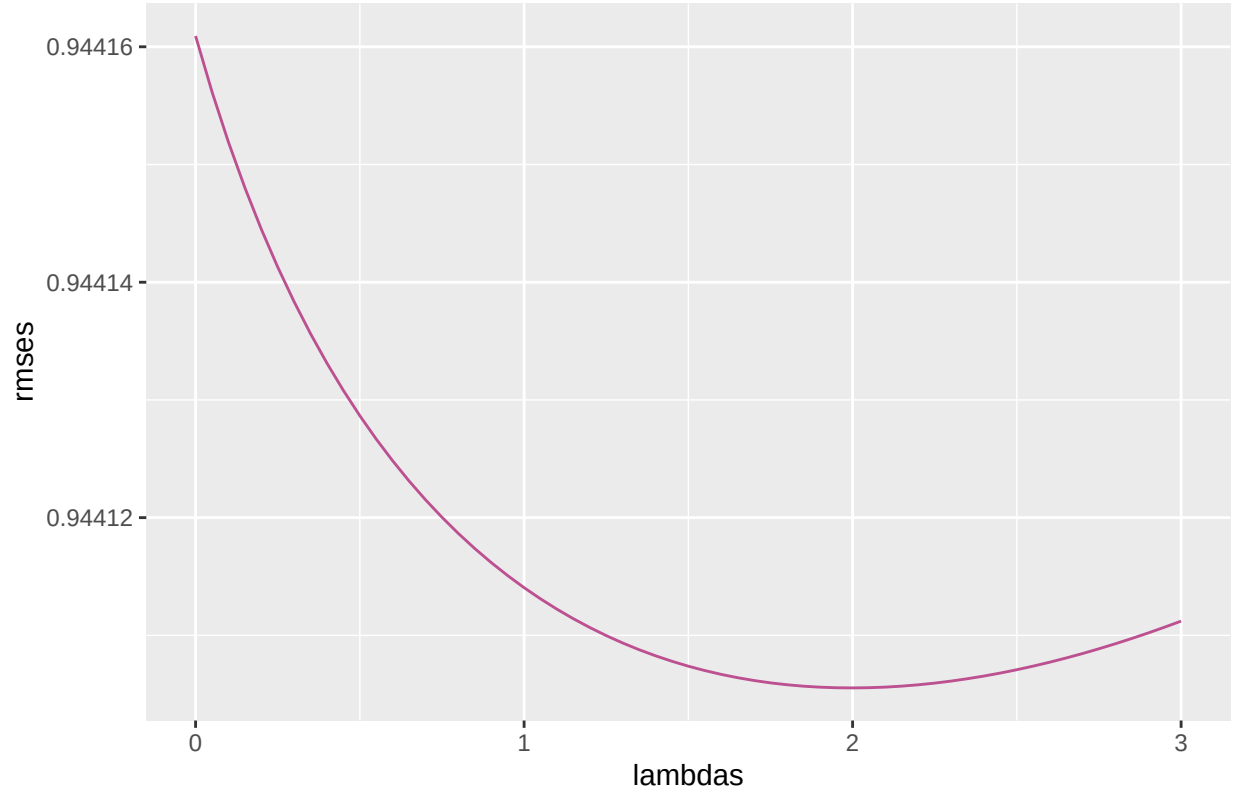
$\hat{b}_i(\lambda) = \frac{1}{n_i+\lambda} \sum(Y_i - \mu)$

Model	rmse
User effect model	0.8866609

title	average_rating	number_of_ratings
Aerial, The (La Antena) (2007)	5	1
Along Came Jones (1945)	5	1
Birds of America (2008)	5	1
Blue Light, The (Das Blaue Licht) (1932)	5	1

```
#finding the best lambda for smoothing movie effect
lambdas <- seq(0, 3, 0.05)
rmses <- sapply(lambdas, function(lambda){
  avg_ratings_by_movie$b_i <- avg_ratings_by_movie$sums / (avg_ratings_by_movie$n + lambda)
  left_join(testing, avg_ratings_by_movie, by = "movieId") %>% mutate(pred = mu + b_i) %>%
    summarize(rmse = RMSE(rating, pred)) %>%
    pull(rmse)
})
```

Figure 6: The RMSEs from the lambdas tested



After finding the ideal λ and regularizing b_i we get a new list of the top rated movies. This is more in line with what you would expected (Table 4).

Table 3: Top rated movies

Table 4: Top rated movies after regularization

Now with our new;y regularized b_i we can recalculate b_u with b_i factored in.

```
#adjust b_u to reg. b_i
avg_ratings_by_user <- training_reg %>%
```

title	average_rating	number_of_ratings
Shawshank Redemption, The (1994)	4.455011	22172
Godfather, The (1972)	4.419745	14049
Usual Suspects, The (1995)	4.366324	17232
Schindler's List (1993)	4.359473	18484

Model	rmse
Reg. Movie + User effect model	0.8664013

```
  mutate(x = rating - mu - b_i) %>%
  group_by(userId) %>%
  summarize(b_u = mean(x),
            n = n(),
            sums = sum(x))
```

Reg ALL

If regularizing for b_i improves the RMSE, it could stand to reason that regularizing all the columns being used would improve the predictions. After exploring the data, the factors that will be used are movie_Id, userId, years_since(b_y), release_year(b_r), and genres(b_g). When regularizing all the factors, it will need to be done one-by-one to account for the columns already regularized so that they work together to build an accurate prediction.

$Y_i = \mu + \hat{b_i} + \hat{b_u} + \hat{b_y} + \hat{b_r} + \hat{b_g}$

Results

After getting the model down to an RMSE of 0.865, it is ready to apply to the final holdout test data. Resulting in the final RMSE of **0.8647925**. A significant improvement over the original 1.06

```
Final <- left_join(final_holdout_test, avg_ratings_by_movie, by = "movieId") %>%
  select(-n,-sums) %>%
  left_join(avg_ratings_by_user, by = "userId") %>%
  select(-n,-sums) %>%
  left_join(avg_ratings_by_genre, by = "genres") %>%
  select(-n,-sums) %>%
  left_join(avg_ratings_by_release_year, by  = "release_year") %>%
  select(-n,-sums) %>%
  left_join(avg_ratings_by_rating_time, by  = "years_since") %>%
  select(-n,-sums) %>%
  mutate(pred = mu + b_i + b_u + b_g + b_r + b_y) %>%
  summarize(x = RMSE(rating, pred))
```

Model	rmse
All Regularized model	0.8650495

Model	rmse
Average Rating	1.0603697
Movie effect model	0.9441609
User effect model	0.8866609
Reg. Movie + User effect model	0.8664013
All Regularized model	0.8650495
Final Results	0.8647925

Conclusion

The objective of this project was to create a model that would get an RMSE of 0.86490 or less. This one has exceeded that with 0.8647925. Overall the regularization model had a 22% improvement over the generic average rating method.

It was limited however due to restrictions of time and hardware resources available on a personal computer. More advance algorithms would require more resources that could help account for a variety of factors. Improvements could be made by taking into account similar user's ratings or similar movies(such as a sequel to a previously highly rated movie).

References

T¨Oscher, A., & Jahrer, M. (2009). The BigChaos Solution to the Netflix Grand Prize. In *https://www.asc.ohio-state.edu/statistics/statgen/joul_aut2009/BigChaos.pdf*. AT&T Labs - Research. https://www.asc.ohio-state.edu/statistics/statgen/joul_aut2009/BigChaos.pdf

Bobbitt, Z. (2021, May 10). *How to interpret Root Mean Square Error (RMSE)*. Statology. https://www.statology.org/how-to-interpret-rmse/

Irizarry, R. A. (n.d.). *Introduction to data Science*. https://rafalab.dfci.harvard.edu/dsbook/