MovieLens Project

Rohit Ravikumar

2/10/2021

Introduction

The goal of this project is to build a model to predict ratings from different movies and users, built around the MovieLens dataset with 10 million observations.

Starting with the fundamentals established in the HarvardX Data Science: Machine Learning course, we develop the model further by including more effects and incorporating regularization. Through these efforts, we achieve a residual mean square error (RMSE), essentially the average error of a given prediction, of 0.864525.

This model utilizes the following variables (some included in the initial MovieLens dataset, and some created from existing variables):

rating: The value we are trying to predict; the rating left by a user on a movie out of 5 points.
movieID: A unique numerical identifier for each movie.
userID: A unique numerical identifier for each user.
n_ratings: The number of reviews left on each movie.
timestamp: The exact time each review was left.
genres: A collection of the genres to which a movie belongs.
date: The timestamp variable, converted to the useful "Date" format.
yearstamp: The year in which each review was left.
year_released: The year in which each movie was released.
timestamp_round: The date on which a review was left, rounded down to the first of the month.
months_since_first: A measure of the time between a user's first review and the current one (in months).
year_diff: The number of years between the time of the review and the release of the movie (in years).
daystamp: A measure of the day of the week on which the review was left (where Sunday is 1, Monday is 2, and so on.)

Separating the Data

We begin by separating the data:

```
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
temp <- movielens[test_index,]
edx <- movielens[-test_index,]

validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)
```

We are left with two separate sets of data: *edx* and *validation*. Going forward, we use edx to test different models, only returning to the validation set at the conclusion of the analysis.

For convenience, we next define a function to calculate the RMSE of a vector of predicted values, something we will return to multiple times throughout the analysis:

```
calc_RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Finally, we separate the *edx* set into a training and a test subset:

```
set.seed(17, sample.kind = "Rounding")
test_index <- createDataPartition(edx$rating, times = 1, p = 0.2, list = FALSE)
temp <- edx[test_index,]
edx_train <- edx[-test_index,]

edx_test <- temp %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")

removed <- anti_join(temp, edx_test)
edx_train <- rbind(edx_train, removed)
rm(temp, removed)
```

Each of the different models will be generated on the larger training subset and tested on the smaller test subset, so as to avoid the possibility of overtraining.

Now we are ready to begin our prediction process.

Naive RMSE Model

For completeness and to establish a baseline, we begin with the "naive RMSE" model introduced in the HarvardX Machine Learning course.

Here, we simply take the mean of all ratings as our prediction for every user and movie. While obviously an inefficient and error-prone model for predicting user reviews, the mean will notably generate a more accurate model (with a smaller RMSE) than any other individual value.

We implement the model here:

```
mu_hat <- mean(edx_train$rating)
naive_rmse <- calc_RMSE(edx_test$rating, mu_hat)
```

method	RMSE
Using the average	1.061022

Our starting RMSE is 1.061: in other words, a prediction made by our model will be more than 1 point off out of 5. This is a relatively inaccurate model; fortunately, there are many steps we can take to improve its accuracy.

Movie and User Effects

We continue with the HarvardX Machine Learning course's model by including effects into the model for each movie and each user.

We can be reasonably certain that not all users and movies have the same distribution of scores. Rather, it should be obvious that certain movies are higher-quality than others, and as a result would tend to get higher ratings. Importantly, it is also true that certain users will tend to give higher ratings overall.

These effects can be computed and included in our model using linear regression. Rather than using the *lm* function in R, computationally intensive for so much data, we take the difference per movie for movies and users to calculate b_i, the movie effect, and b_u, the user effect:

```
movie_avgs <- edx_train %>% group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))
user_avgs <- edx_train %>% left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>% summarize(b_u = mean(rating - mu_hat - b_i))
```

We then take these calculated effects for each movie and each user and add them back into the training and test datasets, generating a predicted vector of ratings in the test dataset and computing the RMSE.

```
edx_train <- edx_train %>% left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId")
edx_test <- edx_test %>% left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>% mutate(pred = mu_hat + b_i + b_u)
rmse_0 <- calc_RMSE(edx_test$pred, edx_test$rating)
```

method	RMSE
Using the average	1.0610225
2 effects	0.8669485

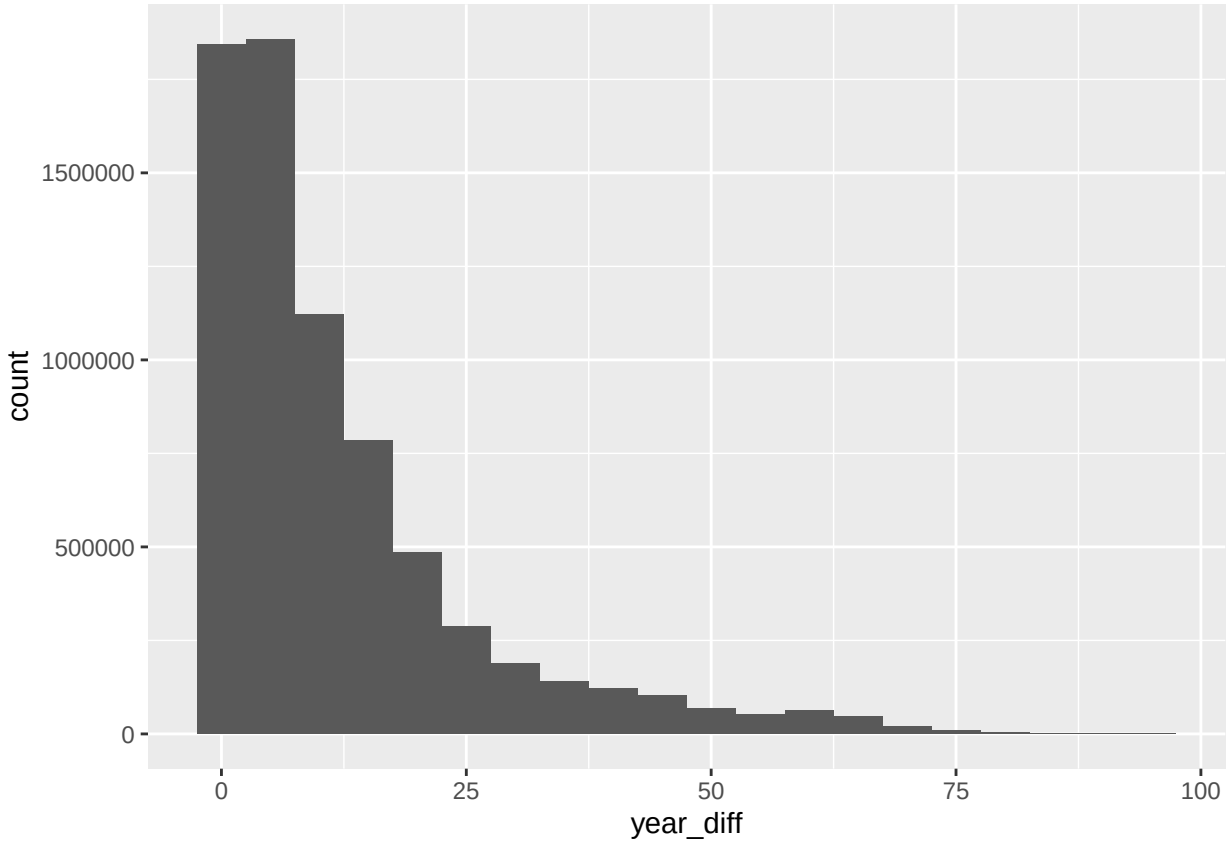
By including the movie and user effects, we see a substantial improvement in our RMSE, going all the way from 1.061 to 0.8669. We will see that by adding more effects into our model, we can improve this result even further.

Year Difference Effect

One effect we can include is the year difference: the number of years between the release of the movie and the submission of the review. It stands to reason that users would tend to approach older movies differently than more recent movies, and this difference can be included in our model to improve its accuracy.

The distribution of year differences can be seen here:

```
edx_train %>% ggplot(aes(year_diff)) + geom_histogram(binwidth = 5)
```



While one would certainly expect most year differences to be clustered in the small range (people are more likely to review movies they just saw), these values are notably distributed across recent and older movies, making this effect a good candidate for improving our model.

Below we calculate a year difference effect, called b_y:

```
year_avgs <- edx_train %>% group_by(year_diff) %>%
  summarize(b_y = mean(rating - mu_hat - b_u - b_i))
```

By adding our b_y values back into our datasets, we are able to calculate an updated prediction vector using b_i, b_u, and b_y:

```
edx_train <- edx_train %>% left_join(year_avgs, by = "year_diff")
edx_test <- edx_test %>% left_join(year_avgs, by = "year_diff") %>%
  mutate(pred_1 = mu_hat + b_i + b_u + b_y)
rmse_1 <- calc_RMSE(edx_test$pred_1, edx_test$rating)
```

method	RMSE
Using the average	1.0610225
2 effects	0.8669485
3 effects	0.8664852

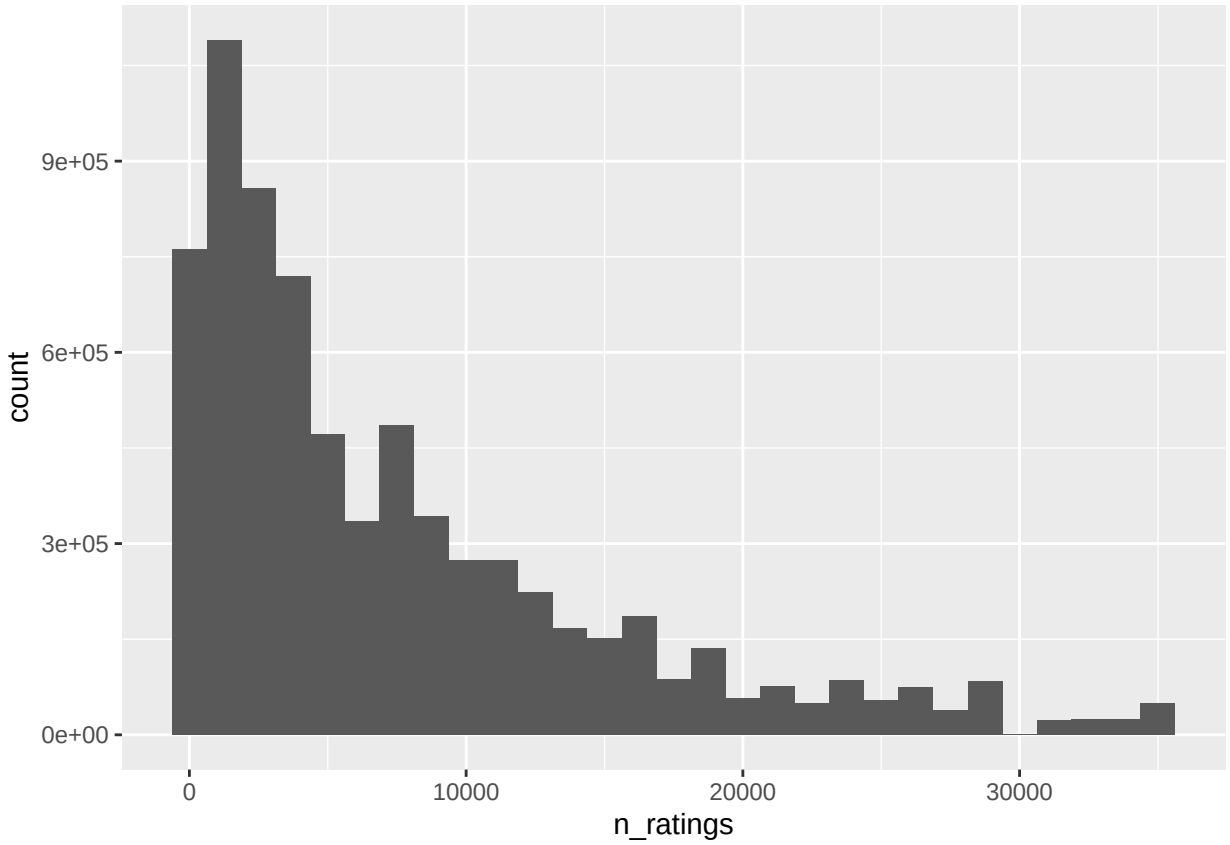
With the inclusion of year effects, our RMSE has improved all the way from 0.8669 to 0.8665.

Number of Ratings Effect

Another effect we can include is the number of ratings per movie. The attention given to a particular movie has a significant effect on the rating of that movie. In particular, it seems unlikely that a movie with many ratings will have a middling score: such movies invite either excitement or derision. This effect can be included in our model with a number of ratings per movie effect, b_n.

We can see the distribution of the number of ratings here:

```
edx_train %>% ggplot(aes(n_ratings)) + geom_histogram(binwidth = 1250)
```



Again, the variance here makes it a good candidate to include as an effect. We generate b_n values, add them back into our data, and generate a new vector of predictors here:

```
n_rating_avgs <- edx_train %>% group_by(n_ratings) %>%
  summarize(b_n = mean(rating - mu_hat - b_u - b_i - b_y))
edx_train <- edx_train %>% left_join(n_rating_avgs, by = "n_ratings")
edx_test <- edx_test %>% left_join(n_rating_avgs, by = "n_ratings") %>%
  mutate(pred_2 = mu_hat + b_i + b_u + b_y + b_n)
rmse_2 <- calc_RMSE(edx_test$pred_2, edx_test$rating)
```

method	RMSE
Using the average	1.0610225
2 effects	0.8669485
3 effects	0.8664852
4 effects	0.8657256

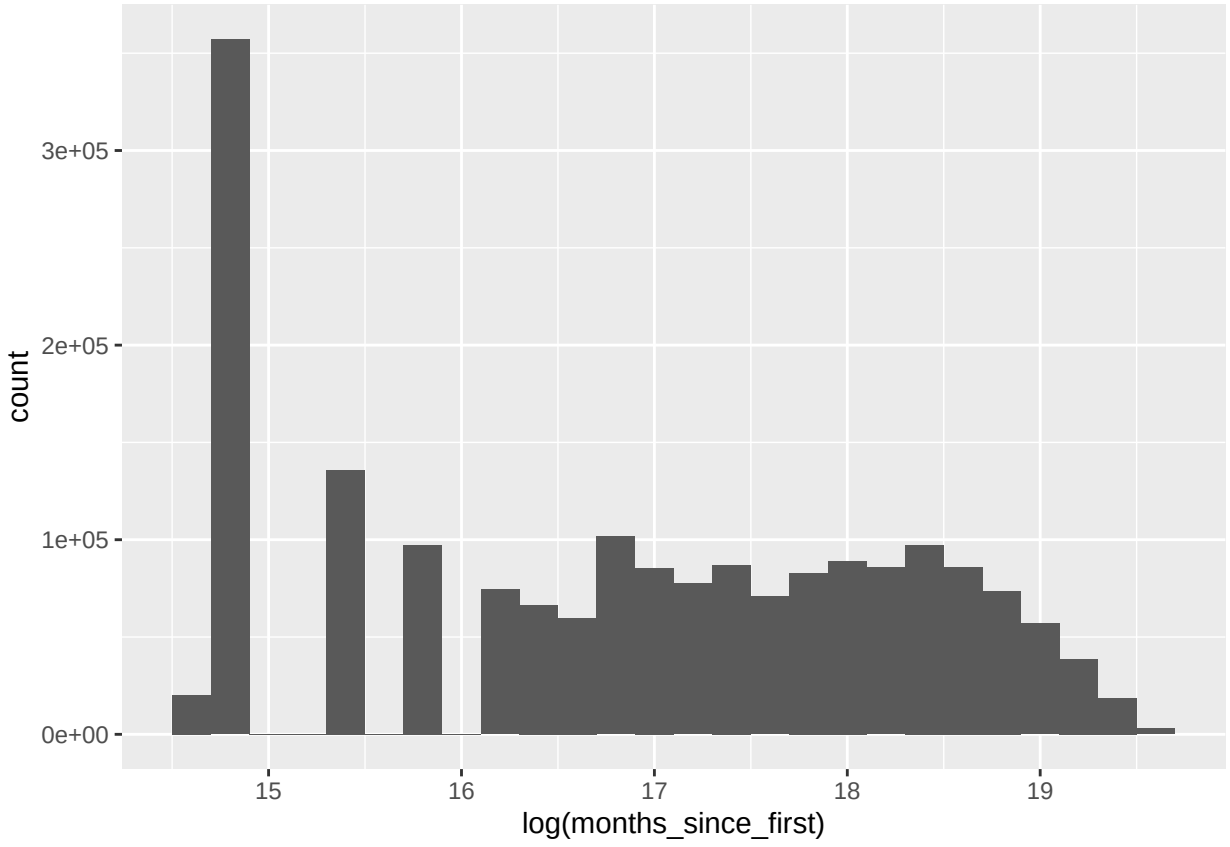
By including the number of ratings with our other effects, our RMSE has dropped to 0.8657.

Months Since First Rating Effect

Next we include an effect for the time between a given rating and the first rating given by that user, b_m. We hypothesize that more time spent rating movies makes users more discerning in their reviews, potentially either increasing or decreasing the rating they tend to give. We measure this value in months.

Here is the distribution of months since first rating: as the raw values are too spread out for a histogram to view them effectively, we instead show here the log transformation:

```
edx_train %>%
  ggplot(aes(log(as.numeric(months_since_first)))) +
  geom_histogram(binwidth = 0.2) +
  xlab("log(months_since_first)")
```



We calculate b_m and generate a new vector of predictors here:

```
since_first_avgs <- edx_train %>% group_by(months_since_first) %>%
  summarize(b_m = mean(rating - mu_hat - b_u - b_i - b_y - b_n))
edx_train <- edx_train %>% left_join(since_first_avgs, by = "months_since_first")
edx_test <- edx_test %>% left_join(since_first_avgs, by = "months_since_first") %>%
  mutate(pred_3 = mu_hat + b_i + b_u + b_y + b_n + b_m)
rmse_3 <- calc_RMSE(edx_test$pred_3, edx_test$rating)
```

method	RMSE
Using the average	1.0610225
2 effects	0.8669485
3 effects	0.8664852
4 effects	0.8657256
5 effects	0.8655298

With the months since first rating effect b_m included, the RMSE has fallen to 0.8655.

Crude Genre Effect

It also stands to reason that the genre of a movie can also be used to predict ratings, as different users will likely have different genre preferences.

Rather than break up movies into individual genres, the MovieLens dataset instead combines all genres a movie belongs to into a single list and stores that as a separate variable. The number of unique genre combinations can be seen here:

```
length(unique(edx_train$genres))
```

```
## [1] 797
```

Nevertheless, even this crude genre measure can be used to calculate a crude genre effect, b_g. We create the b_g values and compute the new RMSE including them.

```
genre_avgs <- edx_train %>% group_by(genres) %>%
  summarize(b_g = mean(rating - mu_hat - b_u - b_i - b_y - b_n - b_m))
edx_train <- edx_train %>% left_join(genre_avgs, by = "genres")
edx_test <- edx_test %>% left_join(genre_avgs, by = "genres") %>%
  mutate(pred_4 = mu_hat + b_i + b_u + b_y + b_n + b_m + b_g)
rmse_4 <- calc_RMSE(edx_test$pred_4, edx_test$rating)
```

method	RMSE
Using the average	1.0610225
2 effects	0.8669485
3 effects	0.8664852
4 effects	0.8657256
5 effects	0.8655298
6 effects	0.8654930

Adding in the crude genre effect has improved our RMSE, though the decrease to 0.86549 is smaller than our decreases thus far. Nevertheless, we continue adding effects.

Day of Week Effect

We now ask a question: Are users likely to leave a more positive or negative review depending on the day of the week? We answer that question with a probability table below (where rows 1-7 reference Saturday-Sunday, respectively).

```
t <- prop.table(table(edx_train$daystamp, edx_train$rating))
knitr::kable(t)
```

0.5	1	1.5	2	2.5	3	3.5	4	4.5	5
0.0012862	0.0051045	0.0016090	0.0107256	0.0052967	0.0315559	0.0122435	0.0394776	0.0081175	0.0210380
0.0013407	0.0060381	0.0017894	0.0127408	0.0055972	0.0370164	0.0133508	0.0454234	0.0089436	0.0245713
0.0016041	0.0061024	0.0019447	0.0124761	0.0059301	0.0370346	0.0145736	0.0453634	0.0096589	0.0243082
0.0015675	0.0057558	0.0017708	0.0116747	0.0056133	0.0349047	0.0134837	0.0419915	0.0089563	0.0222556
0.0012296	0.0053469	0.0017529	0.0110228	0.0052622	0.0330941	0.0119633	0.0394465	0.0079080	0.0209385
0.0013007	0.0053101	0.0016044	0.0106985	0.0049059	0.0326257	0.0118003	0.0394984	0.0078155	0.0213313
0.0011401	0.0047130	0.0014076	0.0097431	0.0043841	0.0294383	0.0105822	0.0363608	0.0071169	0.0200220

Though the differences are subtle, there are noticeable differences across individual ratings for different days of the week: notably, across all ratings, the proportion seems to decline much more on weekends for movies rated 4 than other ratings. As such, we may benefit from including the day of week effect, b_d, in our model.

```
day_avgs <- edx_train %>% group_by(daystamp) %>%
  summarize(b_d = mean(rating - mu_hat - b_u - b_i - b_y - b_n - b_m))
edx_train <- edx_train %>% left_join(day_avgs, by = "daystamp")
edx_test <- edx_test %>% left_join(day_avgs, by = "daystamp")  %>%
  mutate(pred_5 = mu_hat + b_i + b_u + b_y + b_n + b_m + b_g + b_d)
rmse_5 <- calc_RMSE(edx_test$pred_5, edx_test$rating)
```

method	RMSE
Using the average	1.0610225
2 effects	0.8669485
3 effects	0.8664852
4 effects	0.8657256
5 effects	0.8655298
6 effects	0.8654930
7 effects	0.8654926

Our RMSE has decreased, but only very slightly (a decrease of only 0.0000004, quite possibly due only to chance). Facing sharp diminishing marginal returns with adding additional effects to the model, we move to a different technique.

Regularizing the Movie Effect

One common problem with analyzing large sets of data can be the overweighting of less frequent observations. The movie effect b_i in particular exhibits this problem. Below are the movies with the ten lowest calculated b_is:

```
movie_titles <- edx %>% select(movieId, title) %>% distinct()
bottom_ten <- movie_avgs %>% left_join(movie_titles, by = "movieId") %>%
  arrange(desc(b_i)) %>% head(10)
knitr::kable(bottom_ten)
```

movieId	b_i	title
5194	1.487544	Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)
33264	1.487544	Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)
42783	1.487544	Shadows of Forgotten Ancestors (1964)
51209	1.487544	Fighting Elegy (Kenka erejii) (1966)
53355	1.487544	Sun Alley (Sonnenallee) (1999)
63808	1.487544	Class, The (Entre les Murs) (2008)
64275	1.487544	Blue Light, The (Das Blaue Licht) (1932)
5849	1.237544	I'm Starting From Three (Ricomincio da Tre) (1981)
4454	1.154210	More (1998)
26073	1.154210	Human Condition III, The (Ningen no joken III) (1961)

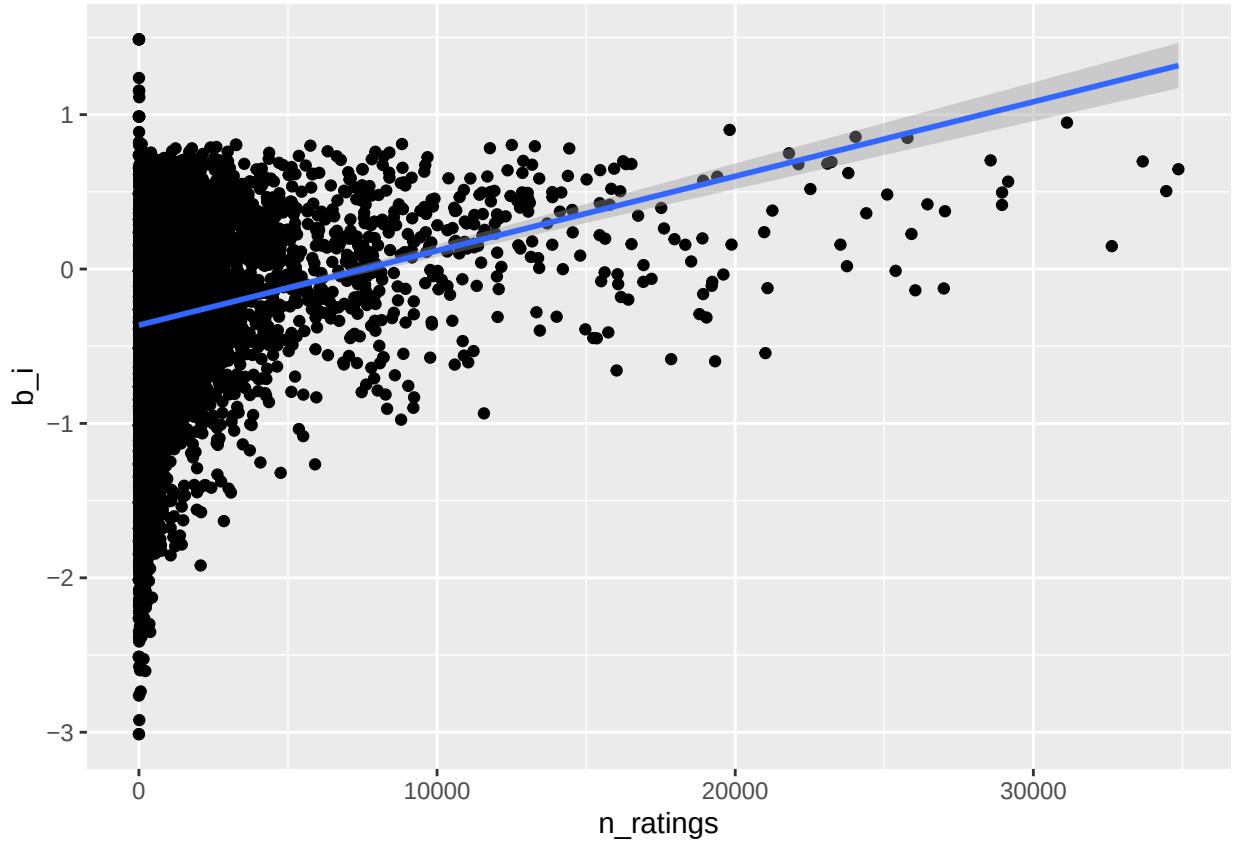
And the ten highest:

```
top_ten <- movie_avgs %>% left_join(movie_titles, by = "movieId") %>%
  arrange(b_i) %>% head(10)
knitr::kable(top_ten)
```

movieId	b_i	title
5805	-3.012456	Besotted (2001)
8394	-3.012456	Hi-Line, The (1999)
61768	-3.012456	Accused (Anklaget) (2005)
7282	-2.921547	Hip Hop Witch, Da (2000)
64999	-2.762456	War of the Worlds 2: The Next Wave (2008)
8859	-2.735861	SuperBabies: Baby Geniuses 2 (2004)
6483	-2.603366	From Justin to Kelly (2003)
61348	-2.598663	Disaster Movie (2008)
63540	-2.574956	Beverly Hills Chihuahua (2008)
6371	-2.525970	PokÃ©mon Heroes (2003)

As you can see, all the above movies are highly obscure. This trend is evident when graphing b_i against the number of ratings a movie has received.

```
edx_train %>% select(b_i, n_ratings) %>% distinct() %>%
  ggplot(aes(n_ratings, b_i)) +
  geom_point() +
  geom_smooth(method = "lm", formula = y ~ x)
```



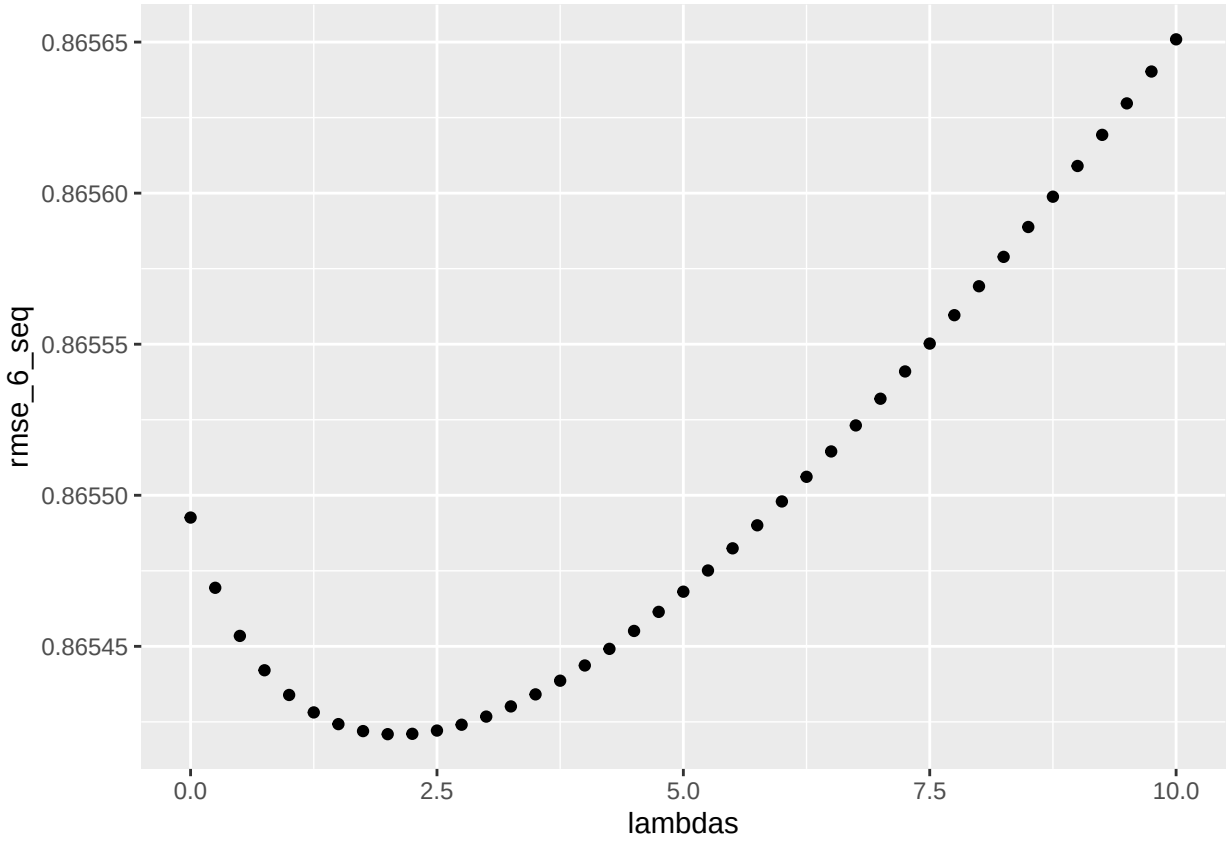
A clear positive trend is present, indicating that movies with fewer ratings, in addition to having more extreme values, also have lower ratings overall. This demonstrates the need to regularize the data by adding a penalty term *lambda* to extreme values.

We apply a regularization process to b_i by trying *lambdas* from 0 to 10 in 0.25 increments, taking the minimum value.

```
lambdas <- seq(0,10,0.25)
rmse_6_seq <- sapply(lambdas, function(x) {
  movie_sums <- edx_train %>% group_by(movieId) %>%
    summarize(s = sum(rating - mu_hat), n_i = n())
  predicted_ratings <- edx_test %>% left_join(movie_sums, by = "movieId") %>%
    mutate(b_i_reg = s / (n_i + x)) %>%
    mutate(pred_reg = mu_hat + b_i_reg + b_u + b_y + b_n + b_m + b_g + b_d) %>% ungroup()
  return(calc_RMSE(predicted_ratings$pred_reg, edx_test$rating))
})
```

We can see the different *lambdas* and their effects on RMSE here:

```
qplot(lambdas, rmse_6_seq)
```



Taking the minimum, our results:

method	RMSE
Using the average	1.0610225
2 effects	0.8669485
3 effects	0.8664852
4 effects	0.8657256
5 effects	0.8655298
6 effects	0.8654930
7 effects	0.8654926
7 effects, reg. movie effect	0.8654209

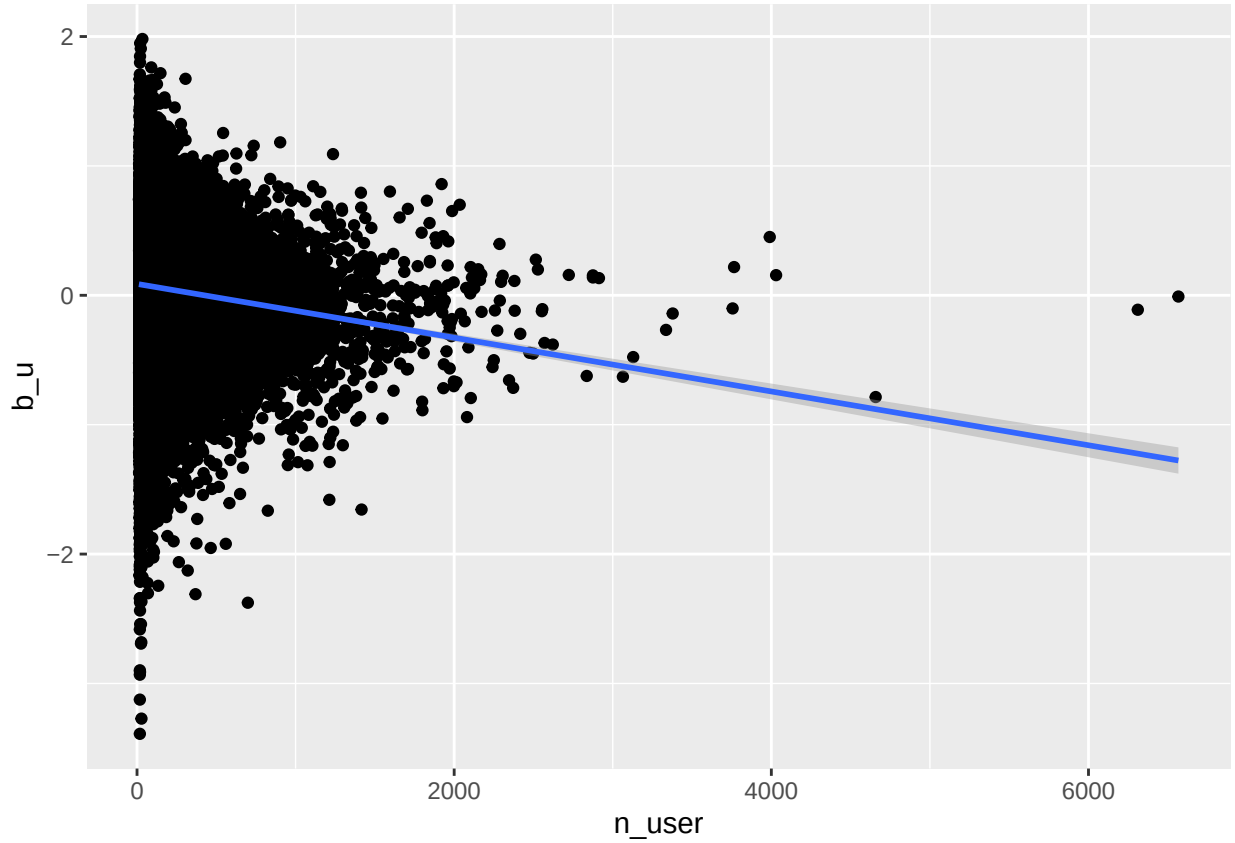
Our decrease in RMSE this time, 0.0000717, while small, is a much larger decrease than our last decrease. This result is encouraging, and we continue with regularization.

Regularizing the User Effect

Similarly to the movie effect, the user effect is also overweighted towards values with few observations. We can see this in this graph of b_u against the number of reviews left by a given user:

```
users <- edx %>% select(userId) %>% distinct()
user_n <- edx %>% group_by(userId) %>% summarize(n_user = n(), userId = mean(userId))
user_avgs %>% inner_join(user_n, by = "userId") %>%
  ggplot(aes(n_user, b_u)) +
```

```
  geom_point() +
  geom_smooth(method = "lm", formula = y ~ x)
```

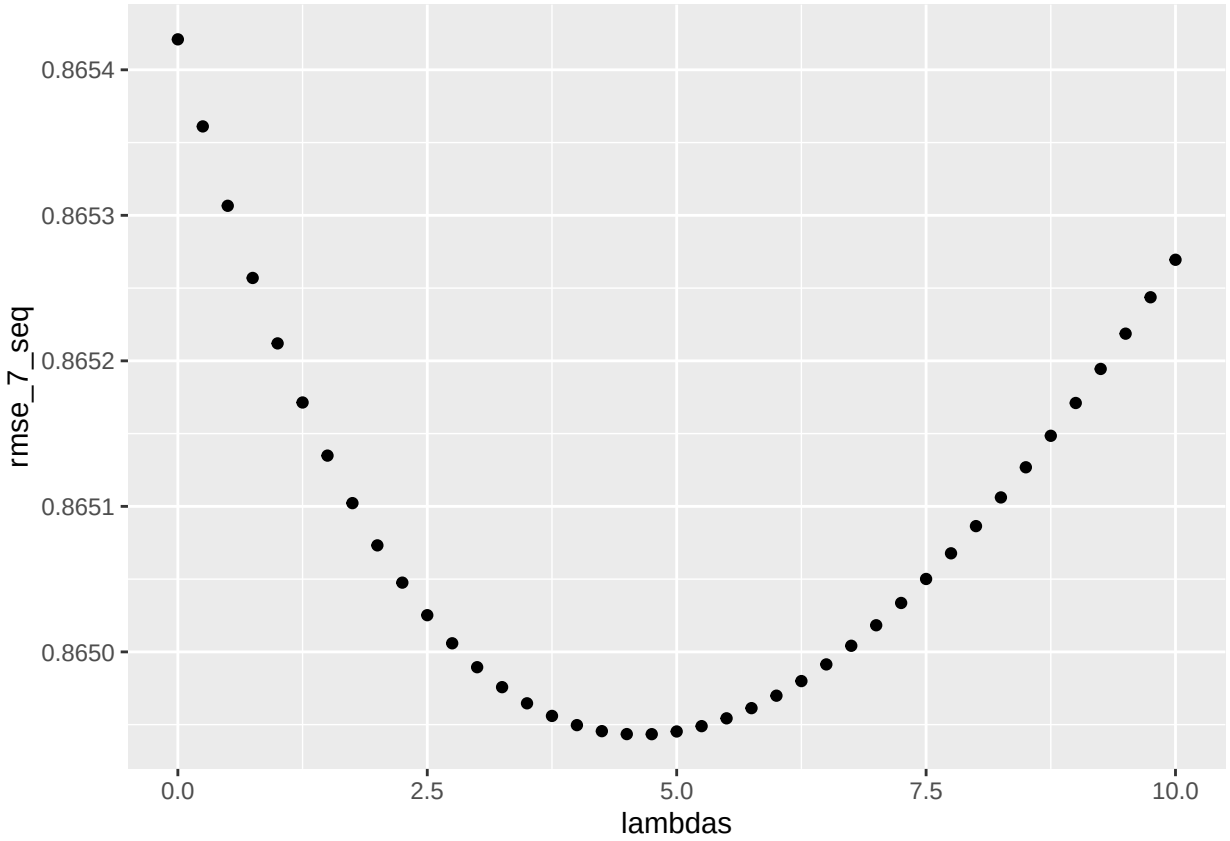


Similarly to our approach with the movie effect, we regularize the user effect b_u by testing *lambdas* from 0 to 10.

```
rmse_7_seq <- sapply(lambdas, function(x) {
  user_sums <- edx_train %>% group_by(userId) %>% summarize(n_u = n())
  predicted_ratings <- edx_test %>% left_join(user_sums, by = "userId") %>%
    mutate(b_u_reg = (b_u*n_u)/(n_u + x)) %>%
    mutate(pred_reg_1 = mu_hat + b_i_reg + b_u_reg + b_y + b_n + b_m + b_g + b_d) %>%
    ungroup()
  return(calc_RMSE(predicted_ratings$pred_reg_1, edx_test$rating))
})
```

We can see the different *lambdas* and their effects on RMSE here:

```
qplot(lambdas, rmse_7_seq)
```



Taking the minimum, our results:

method	RMSE
Using the average	1.0610225
2 effects	0.8669485
3 effects	0.8664852
4 effects	0.8657256
5 effects	0.8655298
6 effects	0.8654930
7 effects	0.8654926
7 effects, reg. movie effect	0.8654209
7 effects, reg. movie/user effects	0.8649435

Our RMSE has fallen all the way to below 0.865, a substantial improvement.

With this improvement, we are ready to test our final model against the validation set.

Final Test Against Validation Data

We add all predicted effects, including the regularized b_i and b_u, into our validation set, and calculate the RMSE here:

```
validation <- validation %>%
  left_join(movie_avgs, by = "movieId") %>%
```

```
  left_join(user_avgs, by = "userId") %>%
  left_join(year_avgs, by = "year_diff") %>%
  left_join(n_rating_avgs, by = "n_ratings") %>%
  left_join(since_first_avgs, by = "months_since_first") %>%
  left_join(genre_avgs, by = "genres") %>%
  left_join(day_avgs, by = "daystamp")  %>%
  left_join(movie_sums, by = "movieId") %>%
  mutate(b_i_reg = s / (n_i + lambda_tune)) %>% ungroup() %>%
  left_join(user_sums, by = "userId") %>%
  mutate(b_u_reg = (b_u*n_u)/(n_u + lambda_tune_1)) %>% ungroup %>%
  mutate(pred_final = mu_hat + b_i_reg + b_u_reg + b_y + b_n + b_m + b_g + b_d)
final_rmse <- calc_RMSE(validation$pred_final, validation$rating)
```

RMSE
0.864525

With an RMSE of 0.864525, this model predicts ratings given movie, user, and time data with a high degree of accuracy.

Conclusion

In a testament to the power of correlated effects, this model was able to bring the RMSE, or average error of a predicted, all the way from 1.061 to 0.864525. Notably, this was without implementing a number of possible interventions. In particular, matrix factorization was not used, and neither was the totality of the genre variable; by breaking each movie down into all its component genres, a more powerful prediction could likely be achieved.

Notably, while this model is a strong fit for this set of data, it does depend on a number of factors to successfully make a prediction: the timestamp of the review and the user's ID and history are all necessary. A more robust model might be able to work with fewer or be able to incorporate different values. Nevertheless, I am pleased with this model, a success for its particular purpose.