Movie Recommendation System

James Musso

10/23/2019

Executive Summary

The goal of this project was to create a movie recommendation system using the 10M version of the MovieLens dataset. This dataset contains 10 million ratings of 10,000 movies by 72,000 users. The basis of the recommendation system is a data-derived algrorithm that can predict, with a reasonable degree of accuracy, the ratings these same users would give to movies they have not previously rated.

To develop such an algorith or predictive model, the MovieLens data was downloaded, explored and imported into an appropriately structured data frame (tidy form) using the RStudio platform. The data was then further manipulated or transformed to facilitate the development of progressively more complex algorithms until the desired level of accuracy was obtained. Testing these algorithms with independent data was achieved by partitioning our single large data frame into subsets.

The specific measure of accuracy used to assess or test our algorithms was the popular **root mean squared error (RMSE)**. The project's ultimate goal was to develop an alogrithm or predictive model with an RMSE less than 0.8649. That goal was achieved as our final model has an RMSE of 0.8635.

Methodology and Analysis

Installing and Loading the Required R Packages

```
if(!require(tidyverse)) install.packages("tidyverse",
                                         repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table",
                                          repos = "http://cran.us.r-project.org")
if(!require(groupdata2)) install.packages("groupdata2",
                                          repos = "http://cran.us.r-project.org")
if(!require(knitr)) install.packages("knitr",
                                     repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate",
                                         repos = "http://cran.us.r-project.org")
if(!require(markdown)) install.packages("markdown",
                                        repos = "http://cran.us.r-project.org")
if(!require(rmarkdown)) install.packages("rmarkdown",
                                         repos = "http://cran.us.r-project.org")
if(!require(tinytex)) install.packages("tinytex",
                                       repos = "http://cran.us.r-project.org")
# tinytex::install_tinytex()  # installs the LaTeX distribution
```

Importing the *movielens* Data

At the outset, we downloaded, decompressed, and then imported the data into a data frame (*movielens*) containing a rating in each row and potential variables (predictors) in columns.

```
# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")
```

Tidying the *movielens* Data Frame

The data, now in the *movielens* data frame, was further transformed, and the data frame itself was reshaped so that each potential predictor had its own column and was in its most useful form.

Specifically, the title of the movie and the year it was released were each given their own column, and the timestamp data was converted into a more understandable format before being split into its component parts. It would now be easier to determine whether ratings were influenced by, for example, the day of the week or the hour of the day the ratings were submitted.

As a preliminary step, we first removed from our *movielens* data frame seven ratings (ranging from 2.0 to 5.0) of a short film ("Pull My Daisy" (1959)) for which no genres were listed. The plot outline and actors suggest the 30-minute movie is a comedy with countercultural themes. But if it became necessary to quantify any effect that genre may have on movies' ratings, it was thought best to remove these seven ratings entirely rather than attempt to impute the missing genres. Given the small number of deletions, this could only have a miniscule impact on our model's predictive accuracy.

```
# Remove seven ratings with no genres, and tidy the *movielens* data frame.

movielens <- movielens %>% filter(genres != "(no genres listed)")

# Add two separate variables, "title" and "movie", from movielens$title.

movielens <- extract(data = movielens, col = title, into = c("title", "movie_year"),
                     perl = TRUE, regex = "(.*[^\\(\\d{4}\\)])\\((\\d{4})")

# Convert movielens$timestamp to POSIX date-time values.

class(movielens$timestamp) = c('POSIXt', 'POSIXct')
as_datetime(movielens$timestamp)

# Add the column "week_day" from movielens$timestamp.

movielens <- mutate(movielens, week_day = wday(timestamp))

# Round the date-time values in movielens$timestamp to the nearest hour.
```

```
movielens <- mutate(movielens, timestamp_round = round_date(movielens$timestamp,
                                                            unit = "hour"))

# Add two separate variables, "date" and "time", from movielens$timestamp.

movielens <- separate(movielens, timestamp_round, into = c("date", "time"), perl = TRUE,
                sep = "\\s", remove = TRUE, convert = FALSE)

# Add three separate variables, "year", "month", and "month_day" from
# movielens$date.

movielens <- separate(movielens, date, into = c("year", "month", "month_day"),
                perl = TRUE, remove = FALSE, convert = FALSE)

# Remove extra zeros from movielens$time.

movielens <- extract(movielens, time, into = "hour", perl = TRUE, regex = "(^\\d{2})",
                remove = TRUE, convert = FALSE)

# Reorder columns.

movielens <- select(movielens, userId, rating, movieId, title, movie_year, genres,
                    timestamp, date, year, month, month_day, week_day, hour)
```

```
head(movielens)
```

```
##   userId rating movieId                  title movie_year
## 1      1      5     122              Boomerang        1992
## 2      1      5     185               Net, The        1995
## 3      1      5     231          Dumb & Dumber        1994
## 4      1      5     292               Outbreak        1995
## 5      1      5     316               Stargate        1994
## 6      1      5     329 Star Trek: Generations        1994
##                         genres           timestamp       date year month
## 1                Comedy|Romance 1996-08-02 06:24:06 1996-08-02 1996    08
## 2         Action|Crime|Thriller 1996-08-02 05:58:45 1996-08-02 1996    08
## 3                       Comedy 1996-08-02 05:56:32 1996-08-02 1996    08
## 4  Action|Drama|Sci-Fi|Thriller 1996-08-02 05:57:01 1996-08-02 1996    08
## 5       Action|Adventure|Sci-Fi 1996-08-02 05:56:32 1996-08-02 1996    08
## 6 Action|Adventure|Drama|Sci-Fi 1996-08-02 05:56:32 1996-08-02 1996    08
##   month_day week_day hour
## 1        02        6   06
## 2        02        6   06
## 3        02        6   06
## 4        02        6   06
## 5        02        6   06
## 6        02        6   06
```

Partitioning the *movielens* Data Frame

The *movielens* data frame was then partitioned into two subsets: a data frame named *edx* to be used for training or developing algorithms, and a data frame named *validation* to be used for testing or assessing the

accuracy of algorithms. Approximately 10% of all the data was used to form the *validation* data frame. One extra step involved ensuring that any userIds, movieIds, and genres in the *validation* data frame were also included in the *edx* data frame.

```
# If using R 3.5 or earlier, use `set.seed(1)`, otherwise ...
# set.seed(1, sample.kind="Rounding")

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

# Make sure userIds, movieIds, and genres in *validation* set are also in *edx* set

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

# Add rows removed from *validation* set back into *edx* set

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

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Developing a Measure of Accuracy

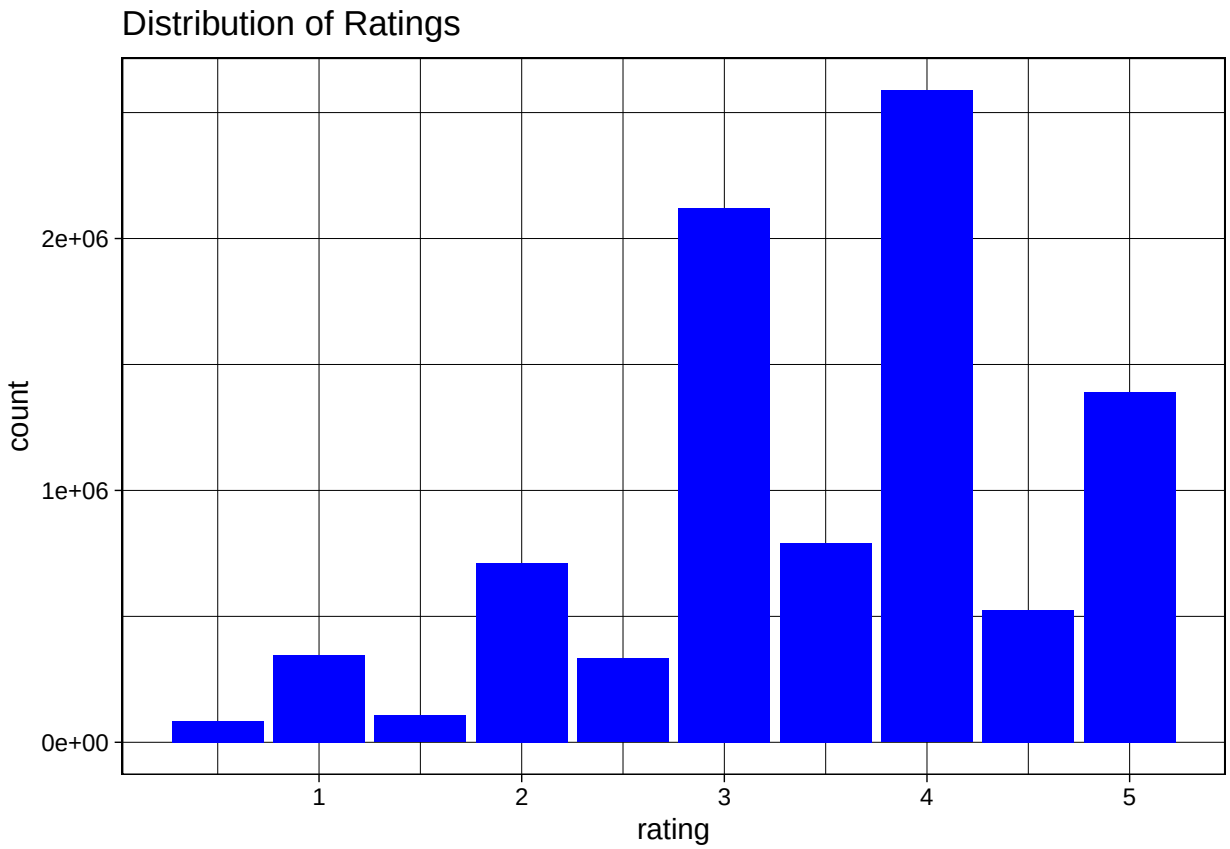
A function was then created to measure the accuracy of our predictive models. The function, named RMSE, measures the root mean squared error of an algorithm. Using this function, we would know whether successive changes to our algorithm were moving us closer to an RMSE of 0.8649, the project goal.

```
RMSE <- function(true_ratings, pred_ratings){
  sqrt(mean((true_ratings - pred_ratings)^2))
}
```

Visualizing the Distribution of Ratings

Now we could begin to think more seriously about predicting ratings. The rating scale spanned zero to five, with intervals of one-half. The ratings in *edx* were plotted to visualize their distribution.

```
ggplot(edx, aes(x=rating)) + geom_bar(fill = "blue") + theme_linedraw() +
  labs(title = "Distribution of Ratings")
```



As could be expected, most ratings fell between three and four, skewing the distribution to the above average side of the rating scale. Still, there was enough variability in the distribution to justify the development of a predictive model.

Developing and Assessing a Random Model

Our starting point was to develop a baseline model that assumes the same rating for all movies and users, with differences explained by random variation. In this scenario, we know that the predicted rating (mu_hat) with the lowest RMSE is the average of all ratings of all movies across all users. We can compute the RMSE of this random model by comparing mu_hat to the actual ratings in the *validation* data frame.

```
mu_hat <- mean(edx$rating)
mu_hat
```

```
## [1] 3.512367
```

```
random_rmse <- RMSE(validation$rating, mu_hat)
random_rmse
```

```
## [1] 1.058959
```

As we would be comparing RMSE values for different predictive models, a table was created to store all the RMSE results, and the RMSE of the random model was added.

```
rmse_results <- data_frame(Method = "Random Model", RMSE = random_rmse)
knitr::kable(rmse_results[1, ], caption = "RMSE Results")
```

Table 1: RMSE Results

Method	RMSE
Random Model	1.058959

With an RMSE above one, the random model left a lot of room for improvement. But how to improve it?

Developing a Movie Effect Model

The most obvious first step was to take into account the movie that was being rated. Here, we know the predicted rating that minimizes the RMSE can be estimated using the average rating for each movie.

To make it easier to identify the effect of the movie predictor, we isolated the movie effect by centering it on the average of all ratings of all movies across all users (mu). Thus, we subtract mu from a movie's rating and then compute the average of those differences for each movie to obtain the true movie effect (b_i).

```
mu <- mean(edx$rating)
mu
```

```
## [1] 3.512367
```

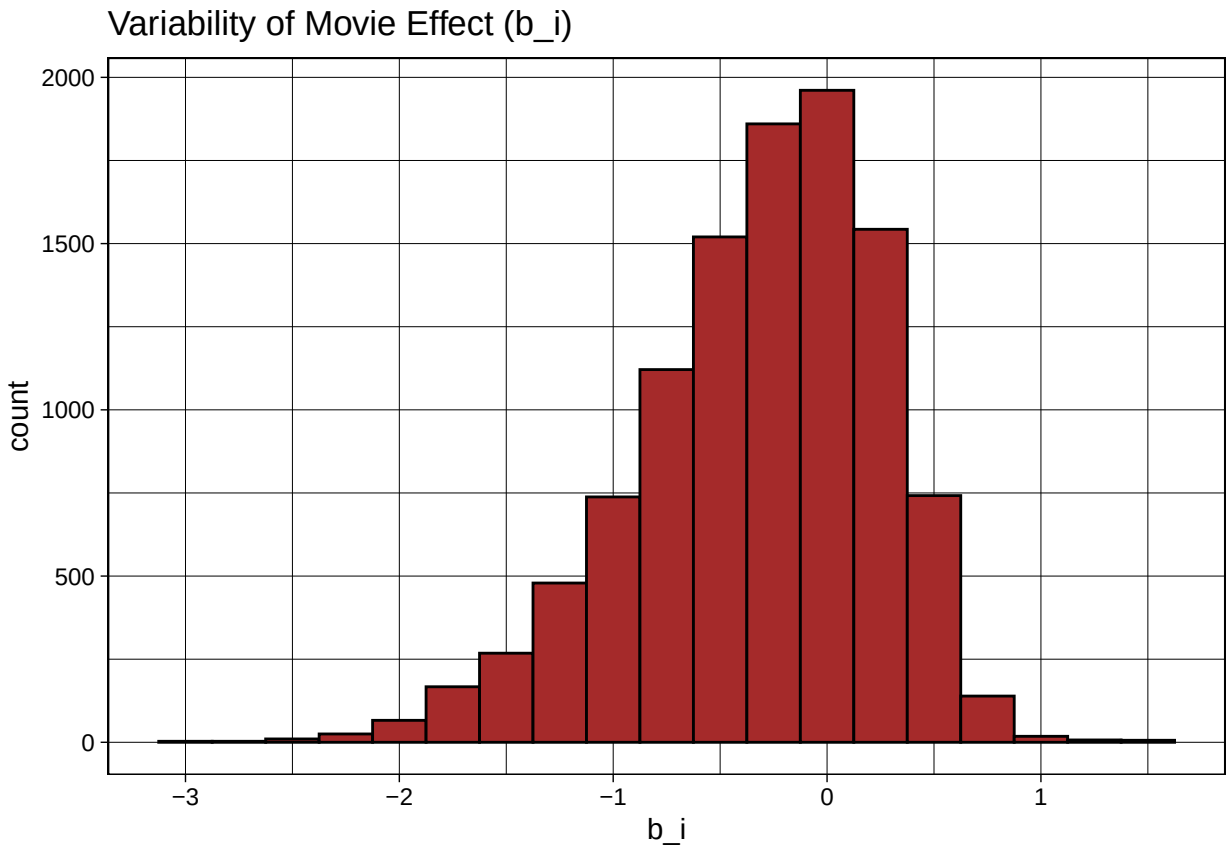
```
movie_avgs <- edx %>% group_by(movieId) %>% summarize(b_i = mean(rating - mu))
knitr::kable(movie_avgs[1:6, ], caption = "movie_avgs (first six rows)")
```

Table 2: movie_avgs (first six rows)

movieId	b_i
1	0.4169020
2	-0.3072915
3	-0.3568434
4	-0.6453190
5	-0.4370217
6	0.3001125

The movie effect (b_i) was then plotted so we could visualize its variability.

```
movie_avgs %>% ggplot(aes(x = b_i)) + geom_histogram(binwidth = 0.25,
                                                     color = "black",
                                                     fill = "brown") +
  theme_linedraw() + labs(title = "Variability of Movie Effect (b_i)")
```



Assessing the Movie Effect Model

Based on the variability of b_i, we expected the movie effect to at least somewhat improve the accuracy of our predictive model. To confirm this, we obtained predicted ratings by adding mu back to b_i for each movie, and then computed the RMSE of the new Movie Effect Model. The results can be seen below.

```
predicted_ratings <- mu +
  validation %>% left_join(movie_avgs, by = "movieId") %>% pull(b_i)
movie_rmse <- RMSE(validation$rating, predicted_ratings)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Movie Effect Model", RMSE = movie_rmse))
knitr::kable(rmse_results[1:2, ], caption = "RMSE Results")
```

Table 3: RMSE Results

Method	RMSE
Random Model	1.0589588
Movie Effect Model	0.9426465

The movie effect improved the predictive ability of our model fairly significantly. But we were still far short of our target RMSE.

Adding a User Effect to the Movie Effect Model

The next most obvious factor to consider was the user providing the rating. For this user effect, we knew that the predicted rating with the lowest RMSE could be estimated using the average rating for each user.

Like the movie effect, the user effect was centered on the average of all ratings of all movies across all users (mu). But it was also centered on the movie effect (b_i). So this time, we subtracted both mu and b_i from a user's rating and then computed the average of those differences for each user to obtain the true user effect (b_u).

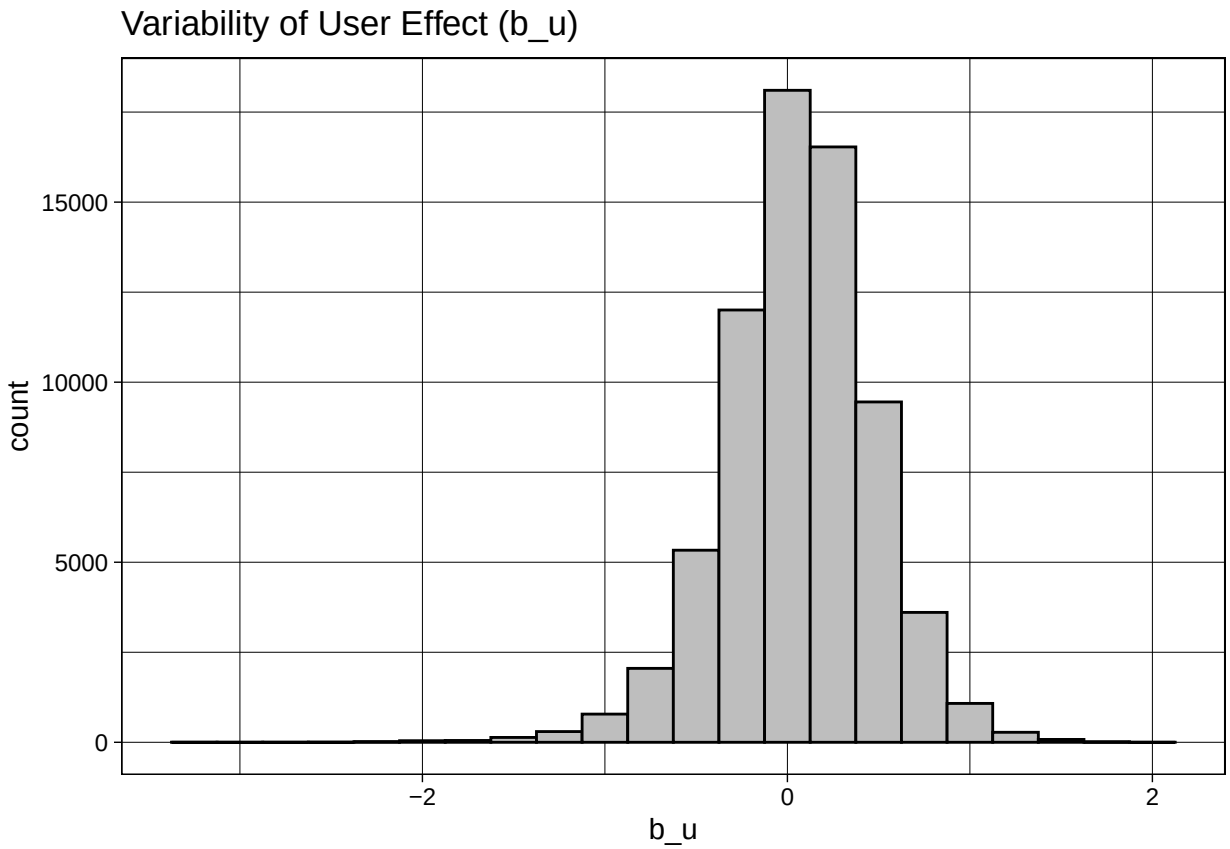
```
user_avgs <- edx %>% left_join(movie_avgs, by = "movieId") %>% group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
knitr::kable(user_avgs[1:6, ], caption = "user_avgs (first six rows)")
```

Table 4: user_avgs (first six rows)

userId	b_u
1	1.7099909
2	-0.3855480
3	0.3062232
4	0.6603918
5	0.0194943
6	0.3229812

The user effect (b_u) was then plotted so we could visualize its variability.

```
user_avgs %>% ggplot(aes(x = b_u)) + geom_histogram(binwidth = 0.25, color = "black",
                                                    fill = "gray") +
  theme_linedraw() + labs(title = "Variability of User Effect (b_u)")
```



Assessing the Movie and User Effects Model

Again, there was enough variability here to suggest that incorporating the user effect in our model would improve its ability to accurately predict ratings. But the RMSE of this Movie and User Effects Model would be the ultimate arbiter on this point. So we added mu and b_i back to b_u to obtain the predicted ratings, which were then compared to the actual ratings in the *validation* data frame by the RMSE function.

```
predicted_ratings <- validation %>% left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>% pull(pred)
user_rmse <- RMSE(validation$rating, predicted_ratings)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Movie + User Effects Model",
                                     RMSE = user_rmse))
knitr::kable(rmse_results[1:3, ], caption = "RMSE Results")
```

Table 5: RMSE Results

Method	RMSE
Random Model	1.0589588
Movie Effect Model	0.9426465
Movie + User Effects Model	0.8642948

Incorporating the user effect in our model proved effective in improving its predictive accuracy. The amount of improvement was not quite as great as that resulting from the movie effect, but it was enough to bring us very close to the project's target RMSE.

Exploring the Data to Further Improve Accuracy

The data underlying the movie and user effects was then further explored to help identify what might be preventing our model from achieving greater accuracy. First we identified the largest prediction mistakes (residuals) based on the movie effect.

```
largest_res_movies <- validation %>% left_join(movie_avgs, by = 'movieId') %>%
  mutate(residual = rating - (mu + b_i)) %>% arrange(desc(abs(residual))) %>%
  select(movieId, title, residual) %>% slice(1:20)
knitr::kable(largest_res_movies,
             caption = "largest_res_movies (20 Largest b_i Residuals)")
```

Table 6: largest_res_movies (20 Largest b_i Residuals)

movieId	title	residual
318	Shawshank Redemption, The	-3.957177
858	Godfather, The	-3.917364
858	Godfather, The	-3.917364
858	Godfather, The	-3.917364
858	Godfather, The	-3.917364
858	Godfather, The	-3.917364
50	Usual Suspects, The	-3.868603
50	Usual Suspects, The	-3.868603
527	Schindler's List	-3.862332
527	Schindler's List	-3.862332
527	Schindler's List	-3.862332
527	Schindler's List	-3.862332
4775	Glitter	3.825373
922	Sunset Blvd. (a.k.a. Sunset Boulevard)	-3.823549
922	Sunset Blvd. (a.k.a. Sunset Boulevard)	-3.823549
912	Casablanca	-3.819412
912	Casablanca	-3.819412
904	Rear Window	-3.814156
904	Rear Window	-3.814156
1212	Third Man, The	-3.813715

Here, the biggest prediction mistakes involved users giving low ratings to some highly acclaimed movies. One possible explanation was that these users' low ratings derived not so much from their feelings about the particular movie, but instead from the type of movie, its subject matter, or genre.

Next, we identified the largest prediction mistakes (residuals) based on the user effect.

```
largest_res_users <- validation %>% left_join(user_avgs, by = 'userId') %>%
  mutate(residual = rating - (mu + b_u)) %>% arrange(desc(abs(residual))) %>%
  select(userId, residual) %>% slice(1:20)
knitr::kable(largest_res_users,
             caption = "largest_res_users (20 Largest b_u Residuals)")
```

Table 7: largest_res_users (20 Largest b_u Residuals)

userId	residual
24193	-4.486912
24193	-4.486912
10364	-4.276612
36022	-4.215888
32463	-4.113976
32463	-4.113976
29796	-4.102736
15922	-4.084463
69802	-4.079748
37651	-4.071232
58361	-4.040321
69672	-4.022227
55534	-4.020865
53192	-4.006524
66451	-4.004572
25932	-3.998288
9759	-3.994252
16221	-3.966465
20931	-3.928661
70858	-3.923029

Here again, the biggest prediction mistakes involved users giving significantly lower ratings than we predicted. In this case we needed to make sure the predictions were based on a sufficient amount of data, and not simply noise. So we computed the number of ratings by each user who appeared among the users with the largest prediction mistakes.

```
edx_ratings_by_user <- edx %>% group_by(userId) %>% mutate(n_ratings_user = n()) %>%
  select(userId, n_ratings_user) %>% distinct(userId, .keep_all = TRUE) %>%
  inner_join(largest_res_users, by = 'userId') %>% arrange(desc(abs(residual)))
knitr::kable(edx_ratings_by_user,
             caption = "edx_ratings_by_user
             (Number of Ratings by Users with Largest Residuals)")
```

Table 8: edx_ratings_by_user (Number of Ratings by Users with Largest Residuals)

userId	n_ratings_user	residual
24193	102	-4.486912
24193	102	-4.486912
10364	17	-4.276612
36022	98	-4.215888
32463	1223	-4.113976
32463	1223	-4.113976
29796	132	-4.102736
15922	180	-4.084463
69802	105	-4.079748
37651	134	-4.071232
58361	23	-4.040321

userId	n_ratings_user	residual
69672	151	-4.022227
55534	39	-4.020865
53192	61	-4.006524
66451	77	-4.004572
25932	23	-3.998288
9759	237	-3.994252
16221	23	-3.966465
20931	180	-3.928661
70858	19	-3.923029

About one-fourth of the largest prediction mistakes could be attributed, at least in part, to having relatively few (fewer than 30) ratings with which to predict a user's behavior.

Exploring the data from a different angle, we next looked at the movies with the highest movie effect.

```
movie_titles <- edx %>% select(movieId, title) %>% distinct()
top_b_i <- movie_avgs %>% left_join(movie_titles, by = 'movieId') %>%
  arrange(desc(b_i)) %>% slice(1:20)
edx_ratings_by_movie <- edx %>% group_by(movieId) %>%
  mutate(n_ratings_movie = n()) %>% select(movieId, n_ratings_movie) %>%
  distinct(movieId, .keep_all = TRUE)
top_b_i <- top_b_i %>% select(movieId, title, b_i) %>%
  left_join(edx_ratings_by_movie, by = 'movieId') %>% arrange(desc(b_i))
top_b_i
```

```
## # A tibble: 20 x 4
##    movieId title                                          b_i n_ratings_movie
##      <dbl> <chr>                                      <dbl>           <int>
##  1    3226 "Hellhounds on My Trail "                  1.49                1
##  2   33264 "Satan's Tango (SÃ¡tÃ¡ntangÃ³) "          1.49                2
##  3   42783 "Shadows of Forgotten Ancestors "         1.49                1
##  4   51209 "Fighting Elegy (Kenka erejii) "          1.49                1
##  5   53355 "Sun Alley (Sonnenallee) "                1.49                1
##  6   64275 "Blue Light, The (Das Blaue Licht) "      1.49                1
##  7    4454 "More "                                   1.24                8
##  8    5194 "Who's Singin' Over There? (a.k.a. Who Si~ 1.24                4
##  9   26048 "Human Condition II, The (Ningen no joken~ 1.24               4
## 10   26073 "Human Condition III, The (Ningen no joke~ 1.24               4
## 11   65001 "Constantine's Sword "                    1.24                2
## 12    5849 "I'm Starting From Three (Ricomincio da T~ 1.15                3
## 13   63808 "Class, The (Entre les Murs) "            1.15                3
## 14   32657 "Man Who Planted Trees, The (Homme qui pl~ 1.06               7
## 15    7452 "Mickey "                                 0.988               1
## 16    7823 "Demon Lover Diary "                      0.988               1
## 17   25975 "Life of Oharu, The (Saikaku ichidai onna~ 0.988               3
## 18   26049 "Fires on the Plain (Nobi) "              0.988               2
## 19   50477 "Testament of Orpheus, The (Testament d'O~ 0.988               1
## 20   53883 "Power of Nightmares: The Rise of the Pol~ 0.988               4
```

The results showed lesser known movies predicted to receive high ratings based on a very limited number of actual ratings. Here was a stronger suggestion that some of our predictions were based on insufficient data that amounted to mere noise.

To complete our examination of the data, we looked at the users with the highest user effect.

```
top_b_u <- user_avgs %>% arrange(desc(b_u)) %>% slice(1:20)
edx_ratings_by_user <- edx %>% group_by(userId) %>% mutate(n_ratings_user = n()) %>%
  select(userId, n_ratings_user) %>% distinct(userId, .keep_all = TRUE)
top_b_u <- top_b_u %>% left_join(edx_ratings_by_user, by = 'userId') %>%
  arrange(desc(b_u))
knitr::kable(top_b_u, caption = "top_b_u (20 Largest User Effects)")
```

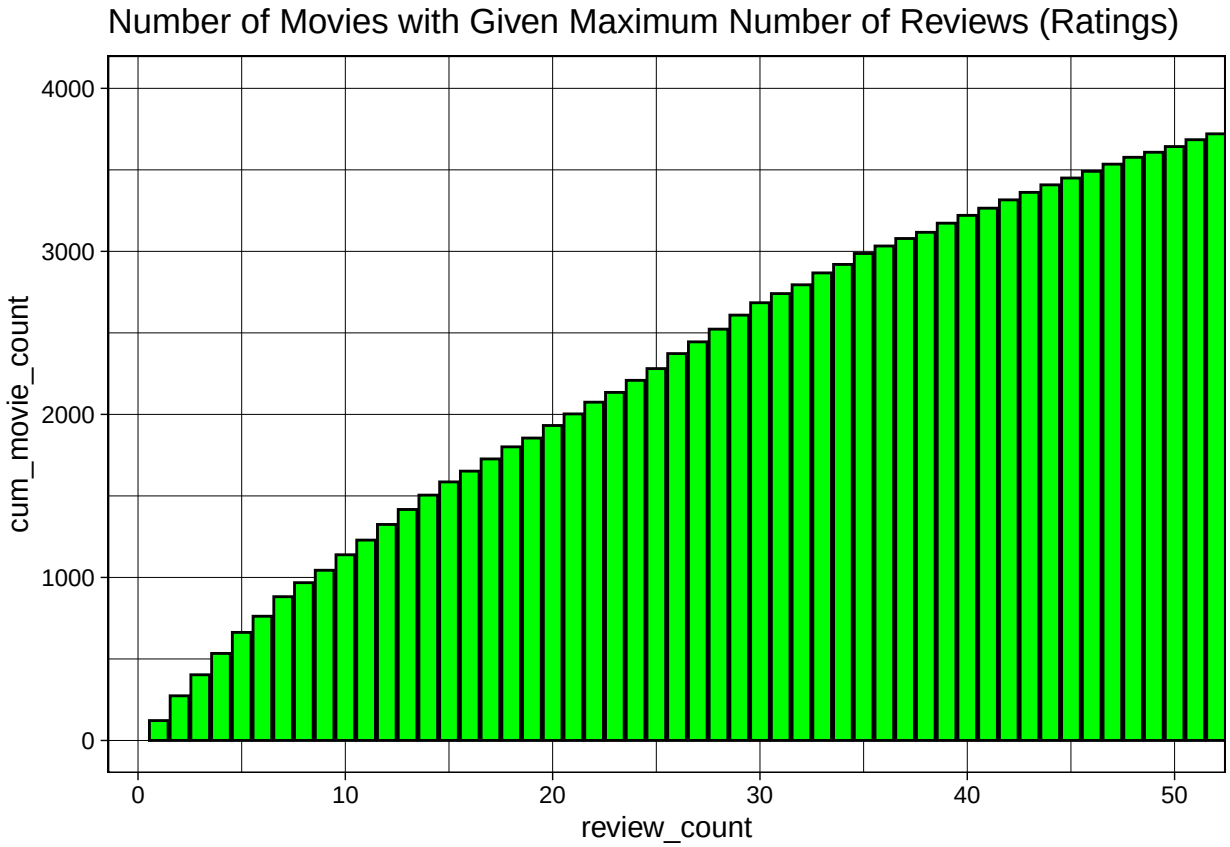
Table 9: top_b_u (20 Largest User Effects)

userId	b_u	n_ratings_user
56965	1.931402	34
13524	1.923585	18
46484	1.816667	23
18591	1.785117	20
45895	1.775110	16
52749	1.763232	91
1943	1.732983	19
36896	1.712935	154
1	1.709991	21
10054	1.706335	76
36022	1.703520	98
46262	1.702857	309
7999	1.673100	37
54009	1.648727	26
67352	1.634589	53
28132	1.629348	36
19813	1.617555	131
42649	1.613857	20
35184	1.611167	22
12330	1.605656	82

The results showed about 40% of these users were predicted to give high ratings based on fewer than 30 actual ratings. This again suggested the possibility that noisy data was impairing our algorithm's ability to accurately predict ratings.

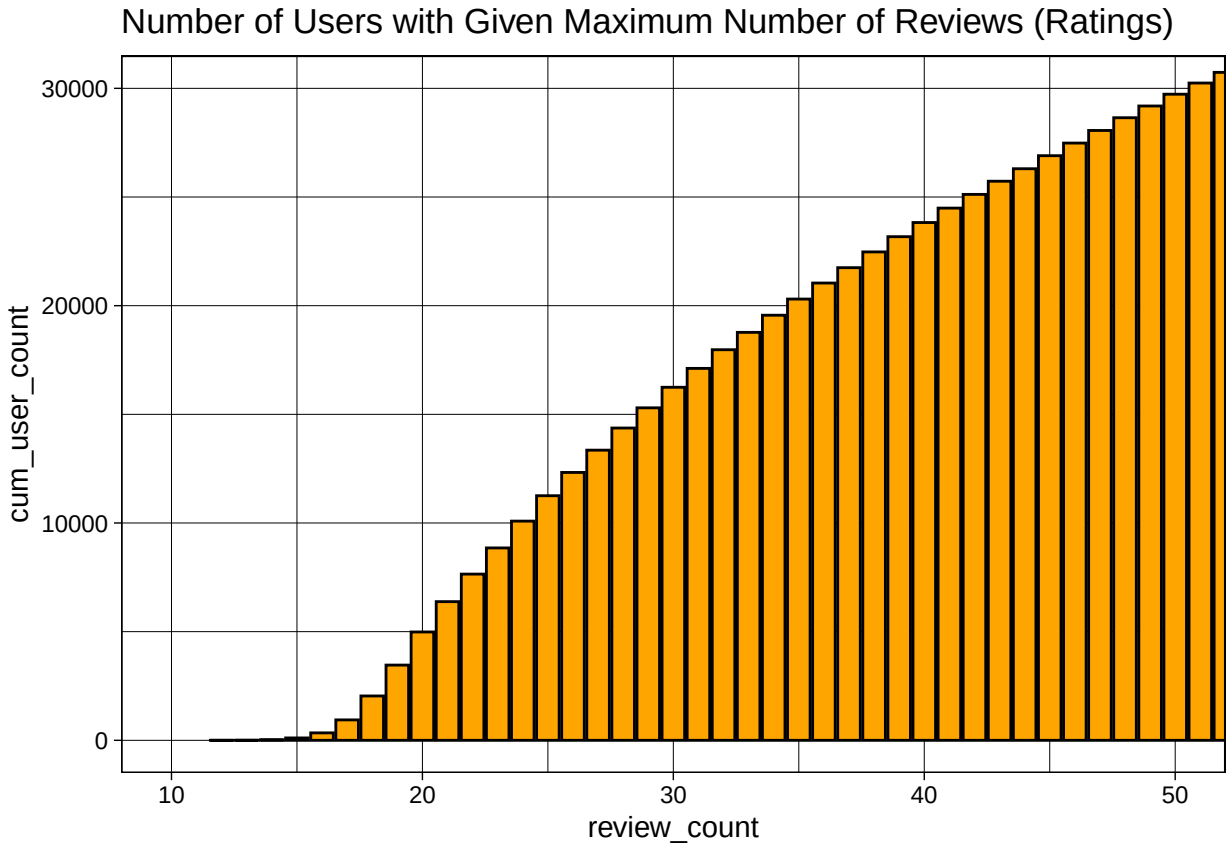
But was this really true? We wanted a more complete picture of the distribution of ratings.

```
# Visualize the distribution of reviews (the number of ratings) by movie.

rev_count_i <- edx %>% group_by(movieId) %>% summarize(review_count = n()) %>%
  arrange(review_count)
rev_count_i <- rev_count_i %>% group_by(review_count) %>%
  summarize(movie_count = n()) %>% arrange(review_count)
rev_count_i <- mutate(rev_count_i, cum_movie_count = cumsum(rev_count_i$movie_count))

ggplot(rev_count_i, aes(x = review_count, y = cum_movie_count)) +
  geom_bar(fill = "green", col = "black", stat = "identity") +
  theme_linedraw() + coord_cartesian(xlim = c(1, 50), ylim = c(1, 4000)) +
  labs(title = "Number of Movies with Given Maximum Number of Reviews (Ratings)")
```



About one-third of all movies received fewer than 50 ratings. As for users...

```
# Visualize the distribution of reviews (the number of ratings) by user.

rev_count_u <- edx %>% group_by(userId) %>%
  summarize(review_count = n()) %>% arrange(review_count)
rev_count_u <- rev_count_u %>% group_by(review_count) %>%
  summarize(user_count = n()) %>% arrange(review_count)
rev_count_u <- mutate(rev_count_u, cum_user_count = cumsum(rev_count_u$user_count))

ggplot(rev_count_u, aes(x = review_count, y = cum_user_count)) +
  geom_bar(fill = "orange", col = "black", stat = "identity") +
  theme_linedraw() + coord_cartesian(xlim = c(10, 50), ylim = c(1, 30000)) +
  labs(title = "Number of Users with Given Maximum Number of Reviews (Ratings)")
```



About 20% of users submitted fewer than 30 ratings. The noise problem appeared to be present for both movies and users.

Accounting for Noise in the Data: Regularization Using Cross-Validation

Accounting for this noisy data might further reduce our RMSE below the project's target level. But how to account for it? A straightforward approach would be to simply remove from our data frames the movies receiving few ratings, and the users submitting few ratings. But removing this much data from our analysis is akin to throwing the baby out with the bath water. The data could be useful later for other purposes.

A more nuanced approach would give less weight to movies and users with fewer ratings. **Regularization**, in effect, does this by penalizing large movie and user effects (that is, large estimates of b_i and b_u) that are based on relatively few ratings (small sample sizes). By regularizing our estimates of b_i and b_u, we can diminish the undue impact of noisy data and further reduce the RMSE of our predictive model.

When b_i and b_u are computed with relatively few ratings, regularization reduces the variability of the effects (b_i and b_u) by applying a penalty term (lambda) as we compute our predictions. The optimal value of lambda is that which minimizes the RMSE of our predictions.

Optimal lambda is best determined by using **cross-validation** to, in effect, test various values of lambda across multiple subsets of data. Most important, these subsets of data are formed from the training dataset (*edx*) rather than the testing dataset(*validation*) to avoid overtraining the model. At the same time, using randomly generated independent subsets of the training dataset to tune a parameter (here, to identify optimal lambda) means we are not using the same dataset that we use to train the whole model.

Creating and Naming Subsets of the *edx* Training Data

Here, we opted to use the **k-fold** approach to cross-validation, with k = 7. That is, we randomly generated seven equal (or near-equal), independent, mutually exclusive samples (folds) from *edx*.

Our first iteration used the first sample (.folds == 1) as a testing or validation subset (val1), and the other six samples were combined to form the training subset (e1). The second iteration used the second sample (.folds == 2) as a validation subset (val2), and the other six samples were combined to form the training subset (e2). The other five iterations were structured in the same way, with each iteration using different data from *edx* to train and validate.

```
# Regularize our estimates of movie (b_i) and user (b_u) effects. Use cross_validation to
# determine the optimal value of lambda in the penalty term of our Regularized Movie
# Effect Model. Then, do the same for our Regularized Movie and User Effects Model. Use
# only the train set (edx) to tune the parameter, lambda.

# Divide the edx training set into 7 equal (or near-equal) parts for k-fold
# cross-validation.

set.seed(1)
folds <- fold(edx, k = 7)

v1 <- folds %>% filter(.folds == 1) %>% as.data.frame()
v2 <- folds %>% filter(.folds == 2) %>% as.data.frame()
v3 <- folds %>% filter(.folds == 3) %>% as.data.frame()
v4 <- folds %>% filter(.folds == 4) %>% as.data.frame()
v5 <- folds %>% filter(.folds == 5) %>% as.data.frame()
v6 <- folds %>% filter(.folds == 6) %>% as.data.frame()
v7 <- folds %>% filter(.folds == 7) %>% as.data.frame()

e1 <- rbind(v2, v3, v4, v5, v6, v7)
e2 <- rbind(v1, v3, v4, v5, v6, v7)
e3 <- rbind(v1, v2, v4, v5, v6, v7)
e4 <- rbind(v1, v2, v3, v5, v6, v7)
e5 <- rbind(v1, v2, v3, v4, v6, v7)
e6 <- rbind(v1, v2, v3, v4, v5, v7)
e7 <- rbind(v1, v2, v3, v4, v5, v6)
```

Having created the new training and testing subsets from *edx*, we needed to make sure, as we did when we originally partitioned the *movielens* data, that each userID, movieId, and genre in the testing subset of an iteration was also in the training subset of that iteration.

```
val1 <- v1 %>%
  semi_join(e1, by = "movieId") %>%
  semi_join(e1, by = "userId") %>%
  semi_join(e1, by = "genres")

val2 <- v2 %>%
  semi_join(e2, by = "movieId") %>%
  semi_join(e2, by = "userId") %>%
  semi_join(e2, by = "genres")

val3 <- v3 %>%
  semi_join(e3, by = "movieId") %>%
```

```
  semi_join(e3, by = "userId") %>%
  semi_join(e3, by = "genres")

val4 <- v4 %>%
  semi_join(e4, by = "movieId") %>%
  semi_join(e4, by = "userId") %>%
  semi_join(e4, by = "genres")

val5 <- v5 %>%
  semi_join(e5, by = "movieId") %>%
  semi_join(e5, by = "userId") %>%
  semi_join(e5, by = "genres")

val6 <- v6 %>%
  semi_join(e6, by = "movieId") %>%
  semi_join(e6, by = "userId") %>%
  semi_join(e6, by = "genres")

val7 <- v7 %>%
  semi_join(e7, by = "movieId") %>%
  semi_join(e7, by = "userId") %>%
  semi_join(e7, by = "genres")

# Add rows removed from validation subsets back into training subsets.

removed <- anti_join(v1, val1)
e1 <- rbind(e1, removed)

removed <- anti_join(v2, val2)
e2 <- rbind(e2, removed)

removed <- anti_join(v3, val3)
e3 <- rbind(e3, removed)

removed <- anti_join(v4, val4)
e4 <- rbind(e4, removed)

removed <- anti_join(v5, val5)
e5 <- rbind(e5, removed)

removed <- anti_join(v6, val6)
e6 <- rbind(e6, removed)

removed <- anti_join(v7, val7)
e7 <- rbind(e7, removed)
```

Identifying the Optimal Value of the Penalty Term (lambda) for our Movie Effect Model

```
# Compute the RMSE for each value of lambda in each iteration.

mu <- mean(edx$rating)
lambdas <- seq(0, 10, 0.25)
```

```
######################## 1

just_the_sum <- e1 %>% group_by(movieId) %>% summarize(s = sum(rating - mu),
                                                       n_i = n())
rmses1 <- sapply(lambdas, function(lambda){
  predicted_ratings <- val1 %>% left_join(just_the_sum, by = 'movieId') %>%
    mutate(b_i = s / (n_i + lambda)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, val1$rating))
})

opt_lambda1 <- lambdas[which.min(rmses1)]
opt_lambda1
```

```
## [1] 2.5
```

```
######################## 2

just_the_sum <- e2 %>% group_by(movieId) %>% summarize(s = sum(rating - mu),
                                                       n_i = n())
rmses2 <- sapply(lambdas, function(lambda){
  predicted_ratings <- val2 %>% left_join(just_the_sum, by = 'movieId') %>%
    mutate(b_i = s / (n_i + lambda)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, val2$rating))
})

opt_lambda2 <- lambdas[which.min(rmses2)]
opt_lambda2
```

```
## [1] 2.25
```

```
######################## 3

just_the_sum <- e3 %>% group_by(movieId) %>% summarize(s = sum(rating - mu),
                                                       n_i = n())
rmses3 <- sapply(lambdas, function(lambda){
  predicted_ratings <- val3 %>% left_join(just_the_sum, by = 'movieId') %>%
    mutate(b_i = s / (n_i + lambda)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, val3$rating))
})

opt_lambda3 <- lambdas[which.min(rmses3)]
opt_lambda3
```

```
## [1] 2.75
```

```
######################## 4

just_the_sum <- e4 %>% group_by(movieId) %>% summarize(s = sum(rating - mu),
                                                       n_i = n())
rmses4 <- sapply(lambdas, function(lambda){
  predicted_ratings <- val4 %>% left_join(just_the_sum, by = 'movieId') %>%
    mutate(b_i = s / (n_i + lambda)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, val4$rating))
})

opt_lambda4 <- lambdas[which.min(rmses4)]
opt_lambda4
```

```
## [1] 2
```

```
######################## 5

just_the_sum <- e5 %>% group_by(movieId) %>% summarize(s = sum(rating - mu),
                                                       n_i = n())
rmses5 <- sapply(lambdas, function(lambda){
  predicted_ratings <- val5 %>% left_join(just_the_sum, by = 'movieId') %>%
    mutate(b_i = s / (n_i + lambda)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, val5$rating))
})

opt_lambda5 <- lambdas[which.min(rmses5)]
opt_lambda5
```

```
## [1] 2.25
```

```
######################## 6

just_the_sum <- e6 %>% group_by(movieId) %>% summarize(s = sum(rating - mu),
                                                       n_i = n())
rmses6 <- sapply(lambdas, function(lambda){
  predicted_ratings <- val6 %>% left_join(just_the_sum, by = 'movieId') %>%
    mutate(b_i = s / (n_i + lambda)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, val6$rating))
})

opt_lambda6 <- lambdas[which.min(rmses6)]
opt_lambda6
```

```
## [1] 1.5
```

```
######################## 7

just_the_sum <- e7 %>% group_by(movieId) %>% summarize(s = sum(rating - mu),
                                                       n_i = n())
rmses7 <- sapply(lambdas, function(lambda){
  predicted_ratings <- val7 %>% left_join(just_the_sum, by = 'movieId') %>%
    mutate(b_i = s / (n_i + lambda)) %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, val7$rating))
})

opt_lambda7 <- lambdas[which.min(rmses7)]
opt_lambda7
```

```
## [1] 2.5
```

At this point, we had identified varying values of lambda that minimized the RMSE of the Movie Effect Model during seven iterations of testing. To determine a single optimal lambda for inclusion in the Movie Effect Model, all that remained to do was compute the mean of these lambdas produced by the seven-fold cross-validation.

```
# Compute the value of optimal lambda for our Regularized Movie Effect Model using the
# results of our k-fold cross-validation.

opt_lambda <- mean(c(opt_lambda1, opt_lambda2, opt_lambda3, opt_lambda4,
                     opt_lambda5, opt_lambda6, opt_lambda7))
opt_lambda
```

```
## [1] 2.25
```

Assessing the Accuracy of the Regularized Movie Effect Model

With the value of optimal lambda in hand, we could now include it in the penalty term of our Regularized Movie Effect Model, and then test the new model with our *validation* test set.

```
# Compute the RMSE of the Regularized Movie Effect Model.

lambda <- opt_lambda
mu <- mean(edx$rating)

movie_reg_avgs <- edx %>% group_by(movieId) %>%
  summarize(b_i = sum(rating - mu) / (n() + lambda), n_i = n())
predicted_ratings <- validation %>%
  left_join(movie_reg_avgs, by = "movieId") %>% mutate(pred = mu + b_i) %>%
  .$pred
reg_movie_rmse <- RMSE(validation$rating, predicted_ratings)

# Update our table of RMSE results

rmse_results <- bind_rows(rmse_results,
```

```
                          data_frame(Method = "Regularized Movie Effect Model",
                          RMSE = reg_movie_rmse))
knitr::kable(rmse_results[1:4, ], caption = "RMSE Results")
```

Table 10: RMSE Results

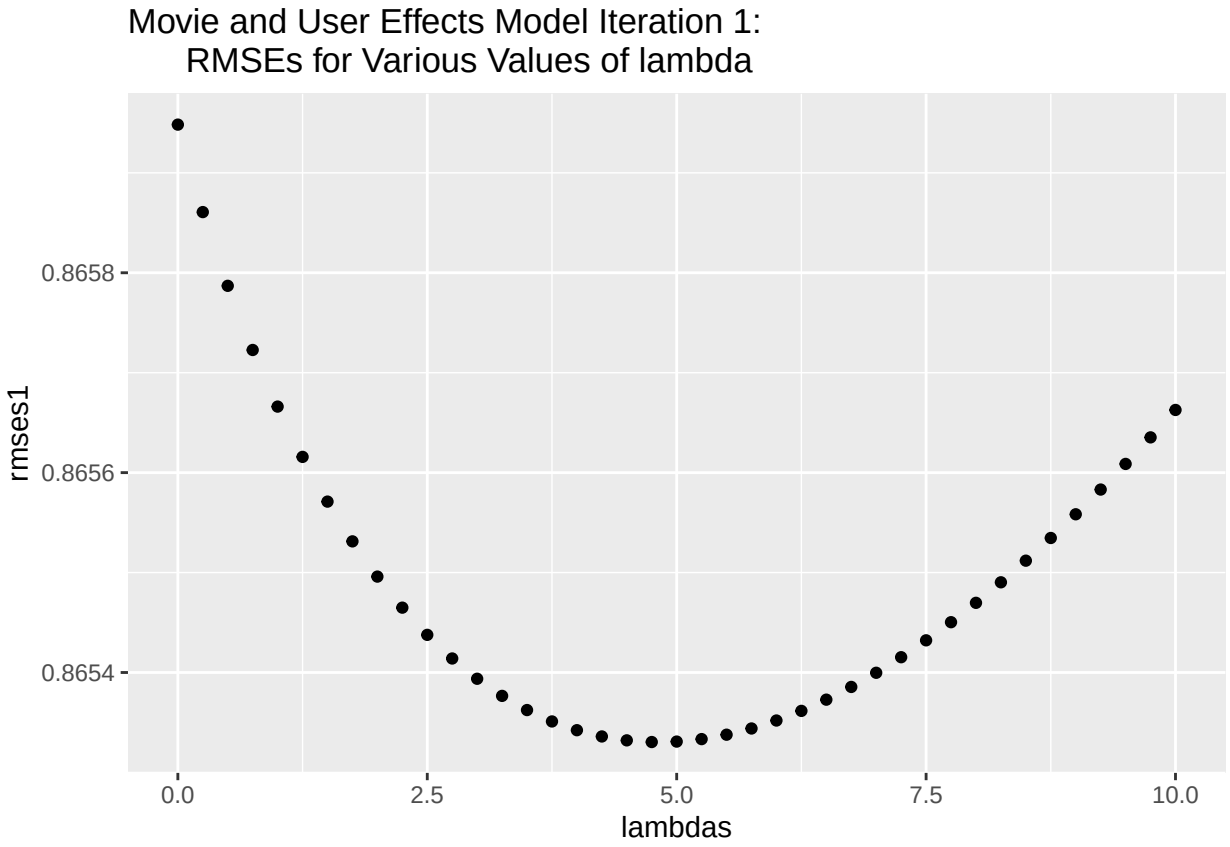
Method	RMSE
Random Model	1.0589588
Movie Effect Model	0.9426465
Movie + User Effects Model	0.8642948
Regularized Movie Effect Model	0.9425910

By regularizing the movie effect (b_i), we were able to achieve a modest reduction in the RMSE of the Movie Effect Model.

Developing a Regularized Movie and User Effects Model

Although the improvement achieved by regularizing the movie effect was slight, it justified using regularization and cross-validation for our entire Movie and User Effects Model. The process involved the same steps used to develop the Regularized Movie Effect Model.

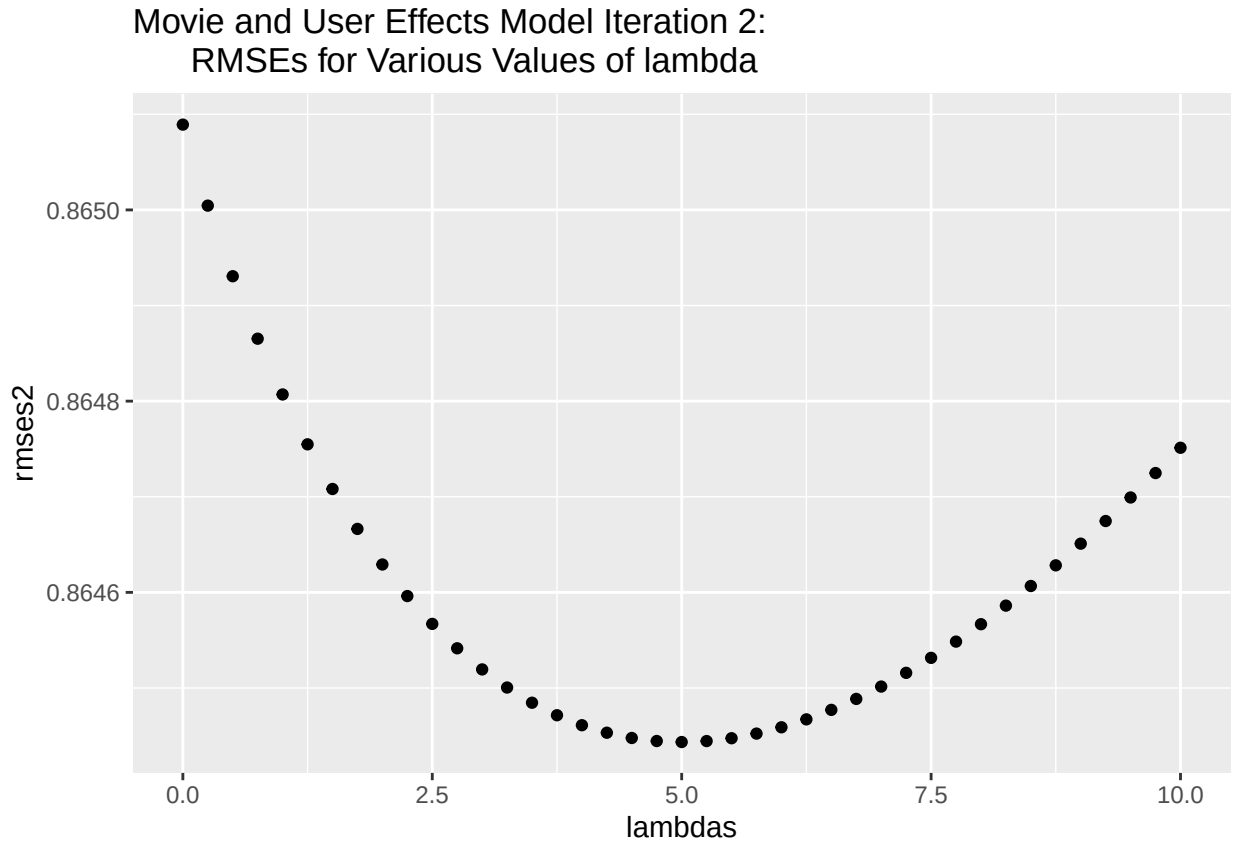
```
# Compute the RMSE for each value of lambda in each iteration.

mu <- mean(edx$rating)
lambdas <- seq(0, 10, 0.25)

########################## 1

rmses1 <- sapply(lambdas, function(lambda){
  b_i <- e1 %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu) / (n() + lambda))
  b_u <- e1 %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu) / (n() + lambda))

  predicted_ratings <- val1 %>% left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(RMSE(val1$rating, predicted_ratings))
})

qplot(lambdas, rmses1,
      main = "Movie and User Effects Model Iteration 1:
      RMSEs for Various Values of lambda")
```




```
opt_lambda1 <- lambdas[which.min(rmses1)]
opt_lambda1
```

```
## [1] 4.75
```

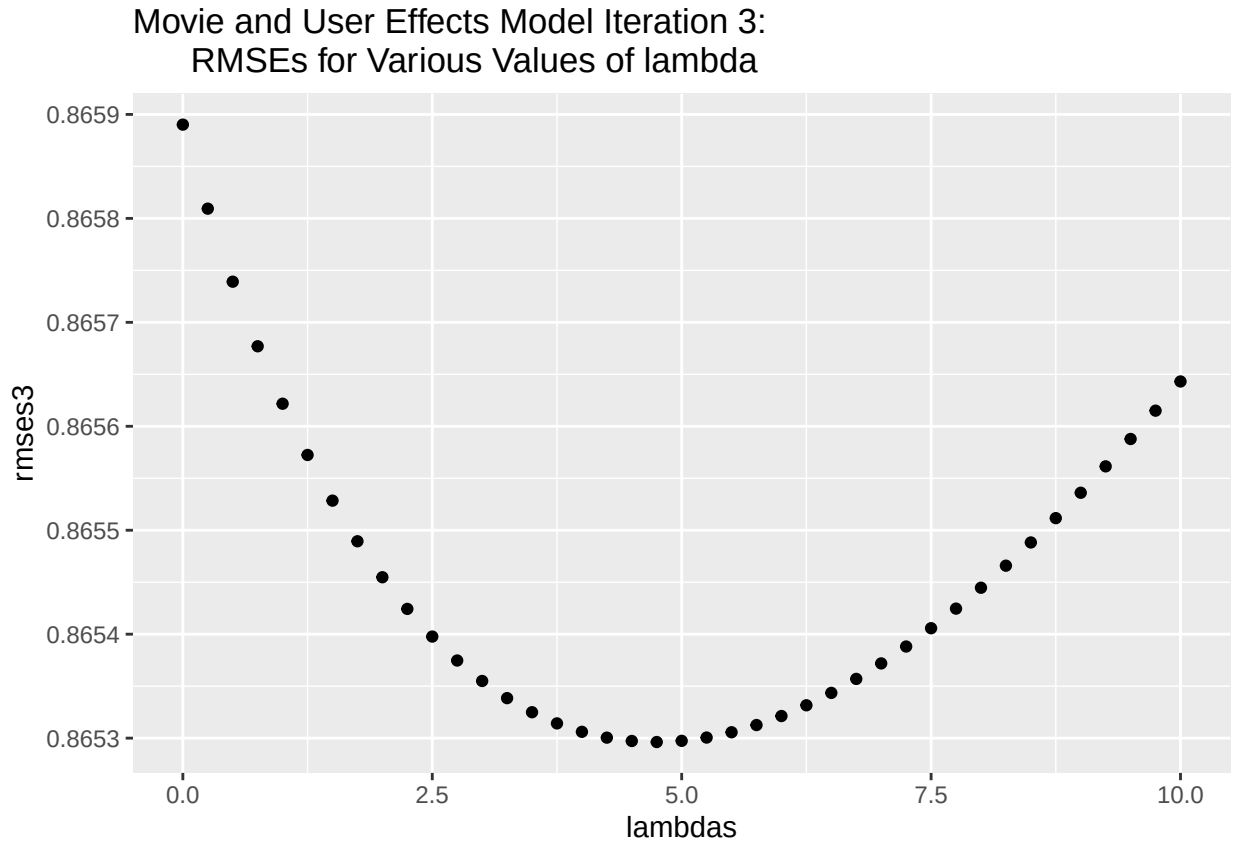
```
######################### 2

rmses2 <- sapply(lambdas, function(lambda){
  b_i <- e2 %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu) / (n() + lambda))
  b_u <- e2 %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu) / (n() + lambda))

  predicted_ratings <- val2 %>% left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(RMSE(val2$rating, predicted_ratings))
})

qplot(lambdas, rmses2,
      main = "Movie and User Effects Model Iteration 2:
      RMSEs for Various Values of lambda")
```




```
opt_lambda2 <- lambdas[which.min(rmses2)]
opt_lambda2
```

```
## [1] 5
```

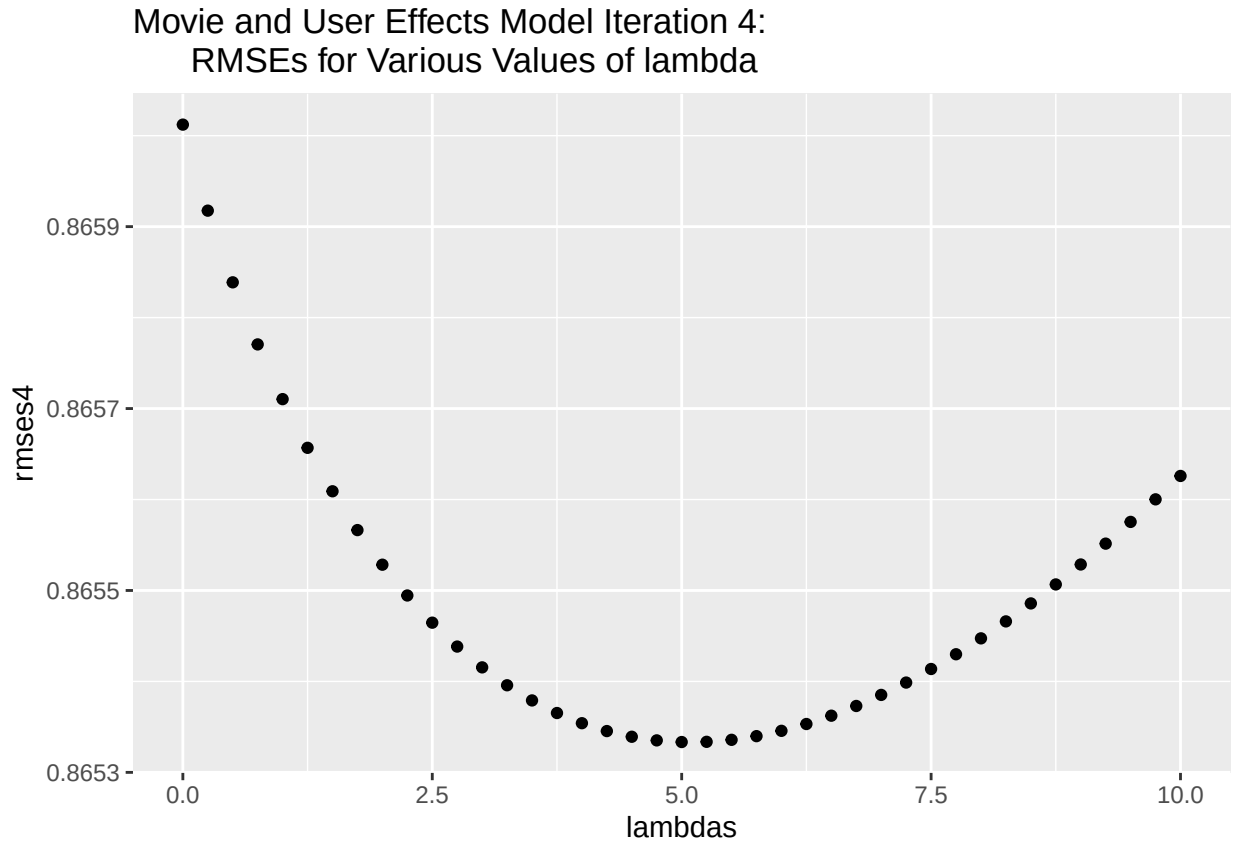
```
######################### 3

rmses3 <- sapply(lambdas, function(lambda){
  b_i <- e3 %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu) / (n() + lambda))
  b_u <- e3 %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu) / (n() + lambda))

  predicted_ratings <- val3 %>% left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(RMSE(val3$rating, predicted_ratings))
})

qplot(lambdas, rmses3,
      main = "Movie and User Effects Model Iteration 3:
      RMSEs for Various Values of lambda")
```




```
opt_lambda3 <- lambdas[which.min(rmses3)]
opt_lambda3
```

```
## [1] 4.75
```

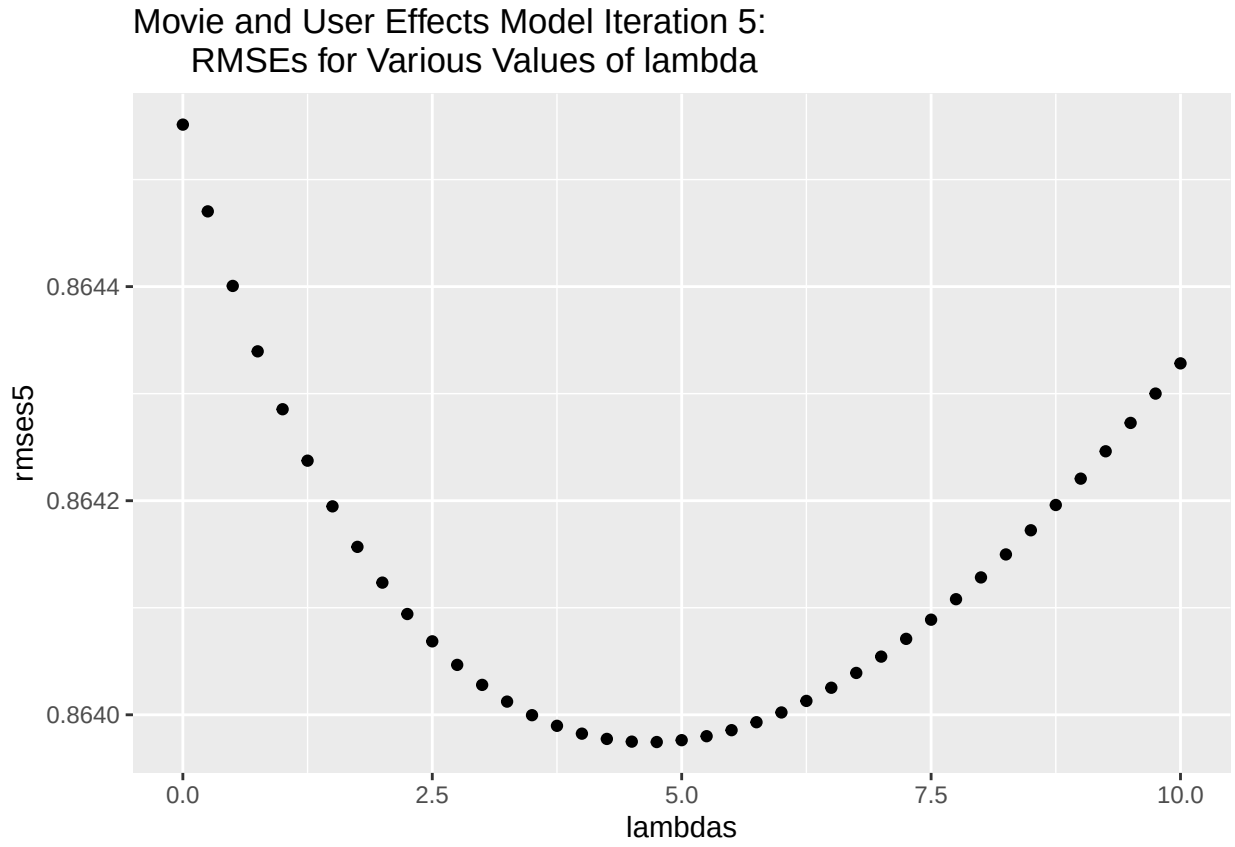
```
######################### 4

rmses4 <- sapply(lambdas, function(lambda){
  b_i <- e4 %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu) / (n() + lambda))
  b_u <- e4 %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu) / (n() + lambda))

  predicted_ratings <- val4 %>% left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(RMSE(val4$rating, predicted_ratings))
})

qplot(lambdas, rmses4,
      main = "Movie and User Effects Model Iteration 4:
      RMSEs for Various Values of lambda")
```




```
opt_lambda4 <- lambdas[which.min(rmses4)]
opt_lambda4
```

```
## [1] 5
```

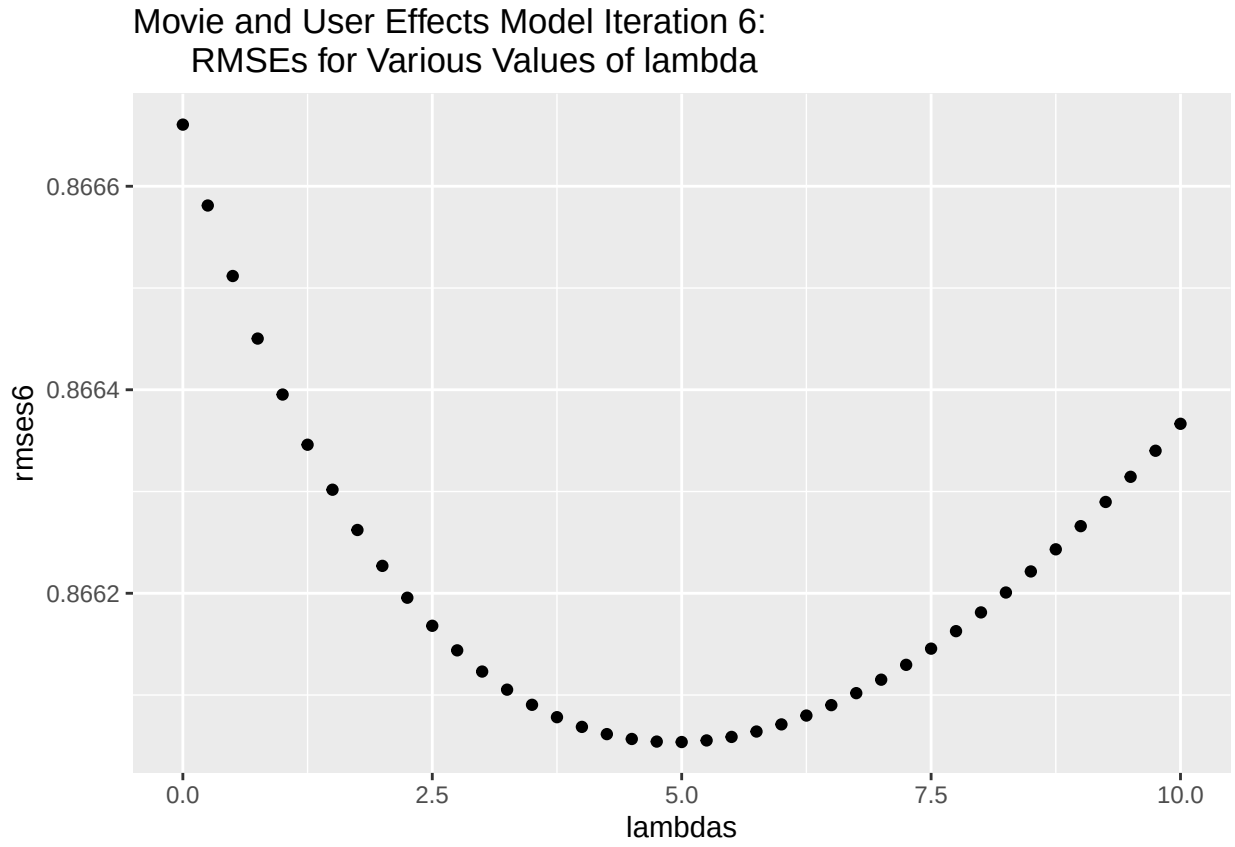
```
########################## 5

rmses5 <- sapply(lambdas, function(lambda){
  b_i <- e5 %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu) / (n() + lambda))
  b_u <- e5 %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu) / (n() + lambda))

  predicted_ratings <- val5 %>% left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(RMSE(val5$rating, predicted_ratings))
})

qplot(lambdas, rmses5,
      main = "Movie and User Effects Model Iteration 5:
      RMSEs for Various Values of lambda")
```




```
opt_lambda5 <- lambdas[which.min(rmses5)]
opt_lambda5
```

```
## [1] 4.75
```

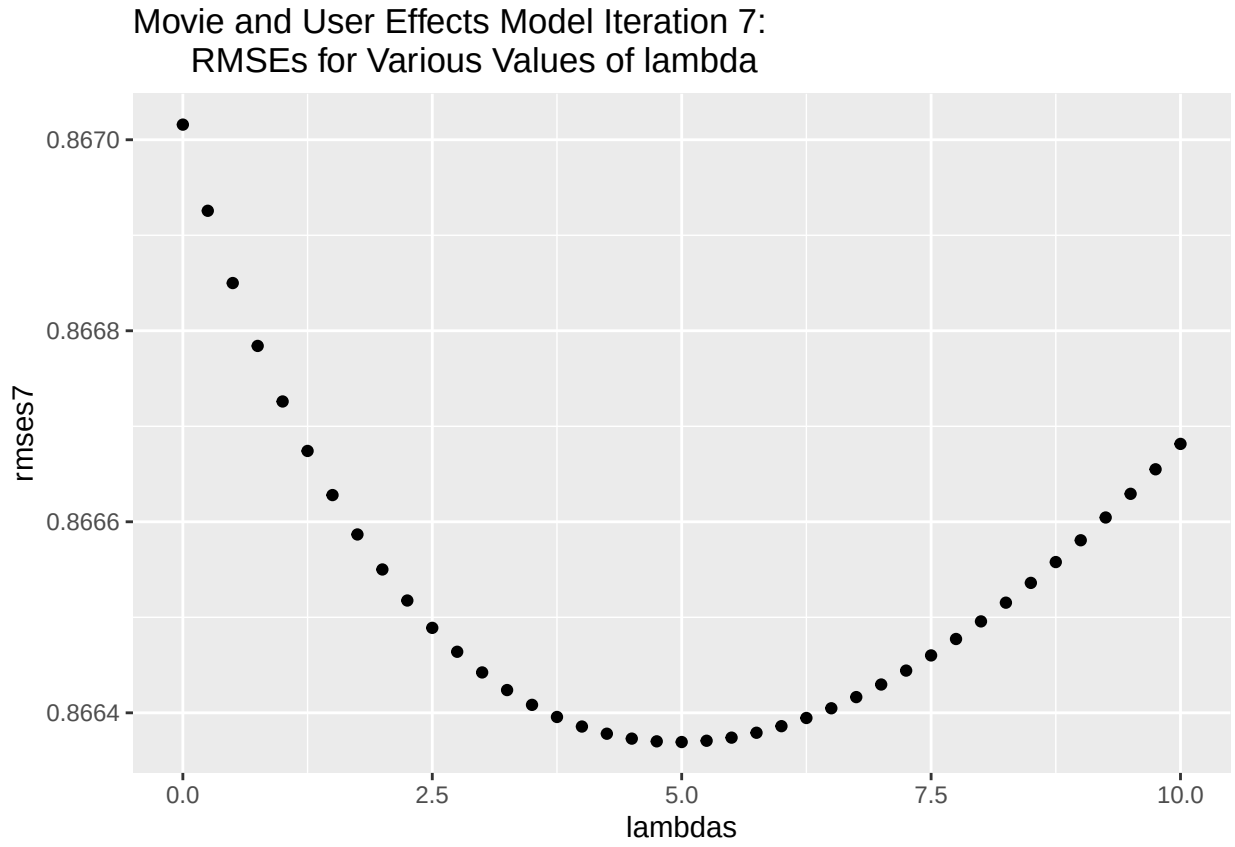
```
######################### 6

rmses6 <- sapply(lambdas, function(lambda){
  b_i <- e6 %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu) / (n() + lambda))
  b_u <- e6 %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu) / (n() + lambda))

  predicted_ratings <- val6 %>% left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(RMSE(val6$rating, predicted_ratings))
})

qplot(lambdas, rmses6,
      main = "Movie and User Effects Model Iteration 6:
      RMSEs for Various Values of lambda")
```




```
opt_lambda6 <- lambdas[which.min(rmses6)]
opt_lambda6
```

```
## [1] 5
```

```
######################### 7

rmses7 <- sapply(lambdas, function(lambda){
  b_i <- e7 %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu) / (n() + lambda))
  b_u <- e7 %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu) / (n() + lambda))

  predicted_ratings <- val7 %>% left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(RMSE(val7$rating, predicted_ratings))
})

qplot(lambdas, rmses7,
      main = "Movie and User Effects Model Iteration 7:
      RMSEs for Various Values of lambda")
```




```
opt_lambda7 <- lambdas[which.min(rmses7)]
opt_lambda7
```

```
## [1] 5
```

```
# Compute the value of optimal lambda for our Regularized Movie and User Effects
# Model using the results of our k-fold cross-validation.

opt_lambda <- mean(c(opt_lambda1, opt_lambda2, opt_lambda3, opt_lambda4, opt_lambda5,
                     opt_lambda6, opt_lambda7))
opt_lambda
```

```
## [1] 4.892857
```

Assessing the Accuracy of the Regularized Movie and User Effects Model

```
# Use optimal lambda to compute the regularized movie and user effects. Then use the
# Regularized Movie and User Effects Model to make predictions. Compute the Model's RMSE.

lambda <- opt_lambda
mu <- mean(edx$rating)

b_i <- edx %>% group_by(movieId) %>%
```

```
  summarize(b_i = sum(rating - mu) / (n() + lambda))
b_u <- edx %>% left_join(b_i, by = 'movieId') %>% group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i) / (n() + lambda))

predicted_ratings <- validation %>% left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = 'userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

reg_movie_user_rmse <- RMSE(validation$rating, predicted_ratings)

# Update our table of RMSE results

rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Regularized Movie and User Effects Model",
                                     RMSE = reg_movie_user_rmse))
knitr::kable(rmse_results[1:5, ], caption = "RMSE Results")
```

Table 11: RMSE Results

Method	RMSE
Random Model	1.0589588
Movie Effect Model	0.9426465
Movie + User Effects Model	0.8642948
Regularized Movie Effect Model	0.9425910
Regularized Movie and User Effects Model	0.8637400

By regularizing both the movie effect (b_i) and the user effect (b_u), we were able to achieve an RMSE of 0.86374, just below the project's target of 0.86490. What other factor could we include in our model to further improve its accuracy?

Exploring the Possibility of a Genre Effect

The list of 20 largest prediction mistakes made by the Movie Effect Model (see Table 6) was dominated by two critically acclaimed movies featuring a large amount of violence and death: The Godfather and Schindler's List. While the former was, strictly speaking, a realistic fictional depiction of gangster life in 20th century New York City, and the latter was a historical drama set during World War II, both films' dark themes evoked powerful emotions. It is easy to imagine some moviegoers having a strong aversion to such movies, regardless of how well they are made.

But to what extent was this characteristic (emotionally powerful with dark subjects) and others like it reflected in the data we had? Was it captured in the genres variable? It was worth exploring.

As with the movie and user effects, we began examining the genre effect (b_g) by computing each genre's average rating and the standard error, as the average rating provides a good estimate of the predicted rating with the lowest RMSE.

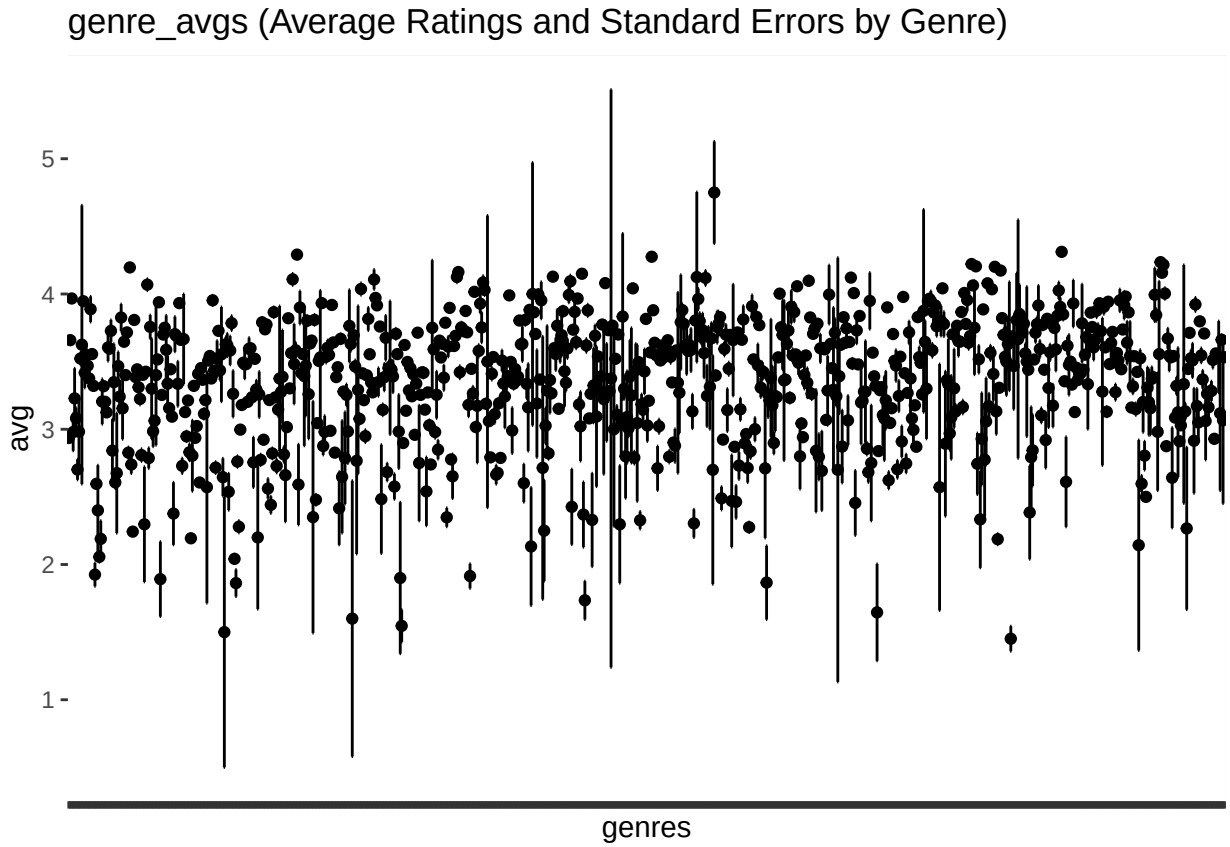
```
genre_avgs <- edx %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), se = sd(rating)/sqrt(n())) %>%
  arrange(desc(se))
knitr::kable(genre_avgs[1:20, ], caption = "genre_avgs
             (Average Ratings and Standard Errors by Genre - 20 Largest SEs)")
```

Table 12: genre_avgs (Average Ratings and Standard Errors by Genre - 20 Largest SEs)

genres	n	avg	se
Adventure|Fantasy|Film-Noir|Mystery|Sci-Fi	4	3.375000	1.0680005
Comedy|Drama|Horror|Sci-Fi	5	2.700000	0.7842194
Adventure|Animation|Musical|Sci-Fi	4	3.500000	0.5400617
Action|Adventure|Animation|Comedy|Sci-Fi	4	3.625000	0.5153882
Action|Drama|Horror|Sci-Fi	5	1.600000	0.5099020
Action|Animation|Comedy|Horror	2	1.500000	0.5000000
Adventure|Comedy|Fantasy|Romance	7	2.714286	0.4862042
Adventure|Comedy|Drama|Fantasy|Mystery|Sci-Fi	9	4.000000	0.4859127
Crime|Drama|Film-Noir|Romance	7	2.571429	0.4555030
Documentary|Romance	3	3.666667	0.4409586
Horror|War|Western	3	3.333333	0.4409586
Action|Adventure|Romance|War	7	2.571429	0.4285714
Action|Crime|Drama|War	10	2.350000	0.4284987
Animation|Horror|IMAX	10	2.700000	0.4229526
Fantasy|Horror|Sci-Fi	7	2.142857	0.3890508
Crime|Documentary	4	3.875000	0.3750000
Adventure|Fantasy|Mystery	15	3.000000	0.3585686
Action|Drama|Musical|Romance	15	2.766667	0.3445724
Animation|Comedy|Drama|Fantasy	16	3.343750	0.3345356
Animation|Documentary|War	4	4.125000	0.3145764

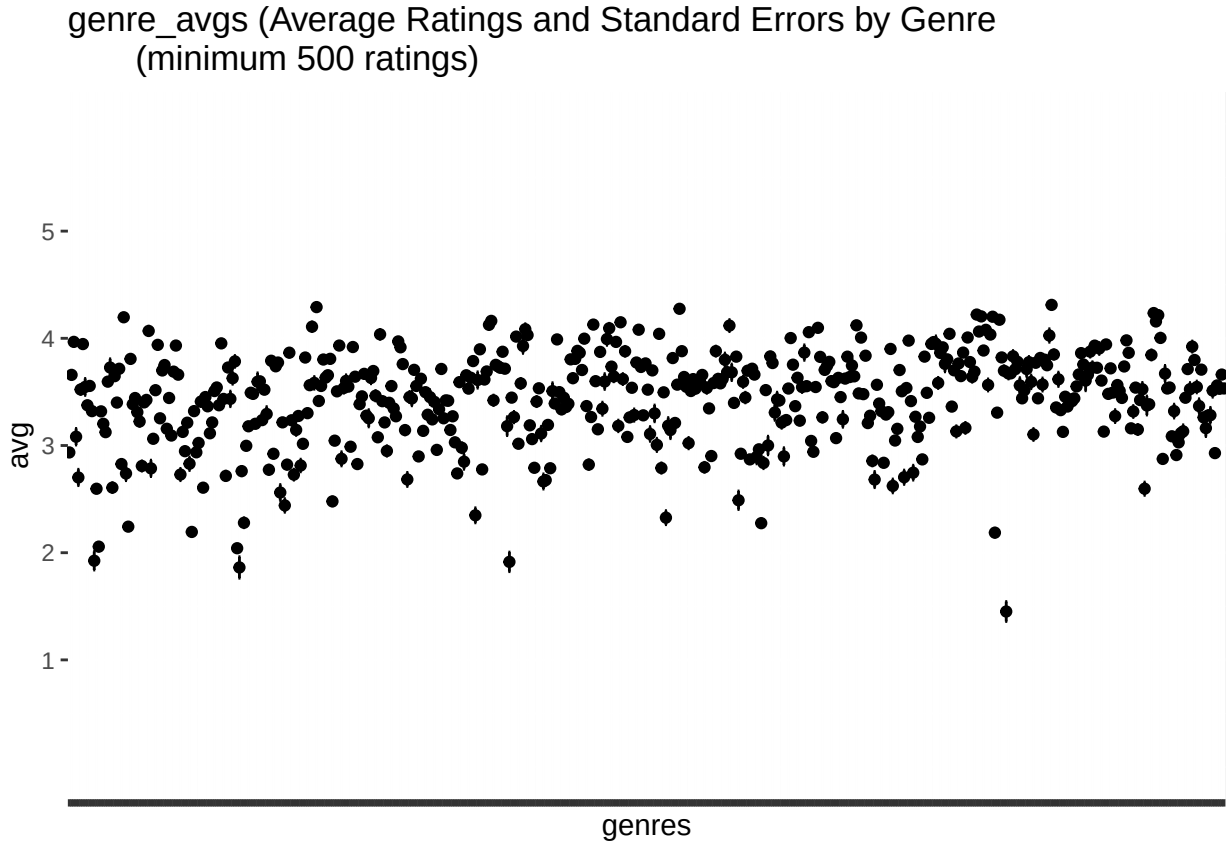
To get a higher-level view of the *genre_avgs* data, we plotted the standard error around each genre average, as well as the variability across all genre averages.

```
genre_avgs %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  scale_x_discrete(labels = NULL) +
  labs(title = "genre_avgs (Average Ratings and Standard Errors by Genre)")
```



Some of the genres' underlying ratings were highly variable (resulting in high standard errors). This was due, at least in part, to the fact that these genres had relatively few ratings (see Table 12). By requiring that each genre have 500 or more ratings, we significantly reduced the standard error around each genre average. When we then plotted variability again, the difference was apparent, suggesting regularization may also be appropriate with respect to the genre effect.

```
genre_avgs <- genre_avgs %>% filter(n >= 500)
genre_avgs %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  scale_x_discrete(labels = NULL) +
  scale_y_continuous(breaks = seq(1, 5, 1),
                     labels = c('1', '2', '3', '4', '5'), limits = c(0, 6)) +
  labs(title = "genre_avgs (Average Ratings and Standard Errors by Genre
       (minimum 500 ratings)")
```



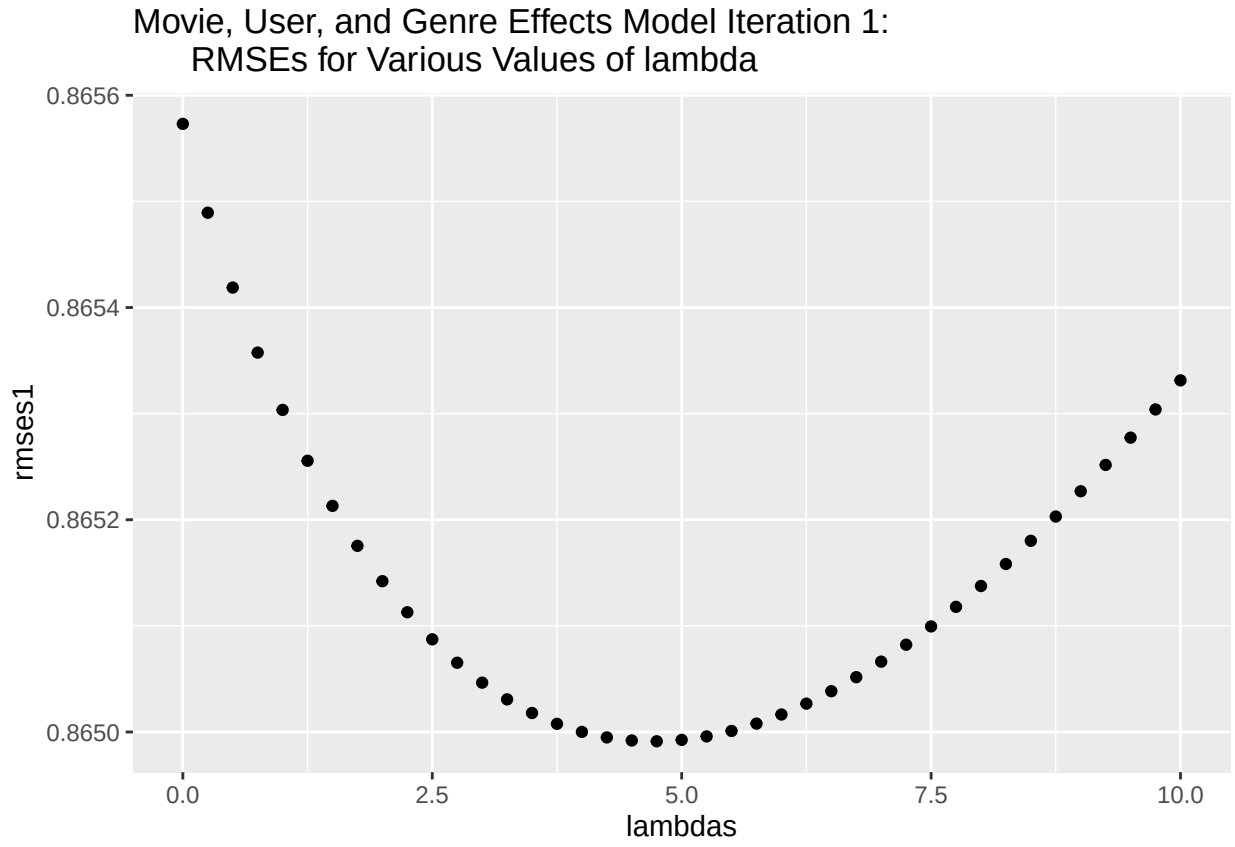
As for the variability of the genre averages across all genres, the results suggested that including genres in our model ought to improve its predictive accuracy.

Developing a Regularized Movie, User, and Genre Effects Model

Since our new model would regularize movie, user, and genre effects, we first needed to identify the optimal value (lambda) of the penalty term. We again used 7-fold cross-validation to achieve this.

```
# Compute the RMSE for each value of lambda in each of the seven iterations.

mu <- mean(edx$rating)
lambdas <- seq(0, 10, 0.25)

############################ 1

rmses1 <- sapply(lambdas, function(lambda){
  b_i <- e1 %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu) / (n() + lambda))
  b_u <- e1 %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i) / (n() + lambda))
  b_g <- e1 %>% left_join(b_i, by = 'movieId') %>% left_join(b_u, by = 'userId') %>%
    group_by(genres) %>% summarize(b_g = sum(rating - mu - b_i - b_u) / (n() + lambda))

  predicted_ratings <- val1 %>% left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    left_join(b_g, by = 'genres') %>%
    mutate(pred = mu + b_i + b_u + b_g) %>% .$pred
```

```
  return(RMSE(val1$rating, predicted_ratings))
})

qplot(lambdas, rmses1,
      main = "Movie, User, and Genre Effects Model Iteration 1:
      RMSEs for Various Values of lambda")
```

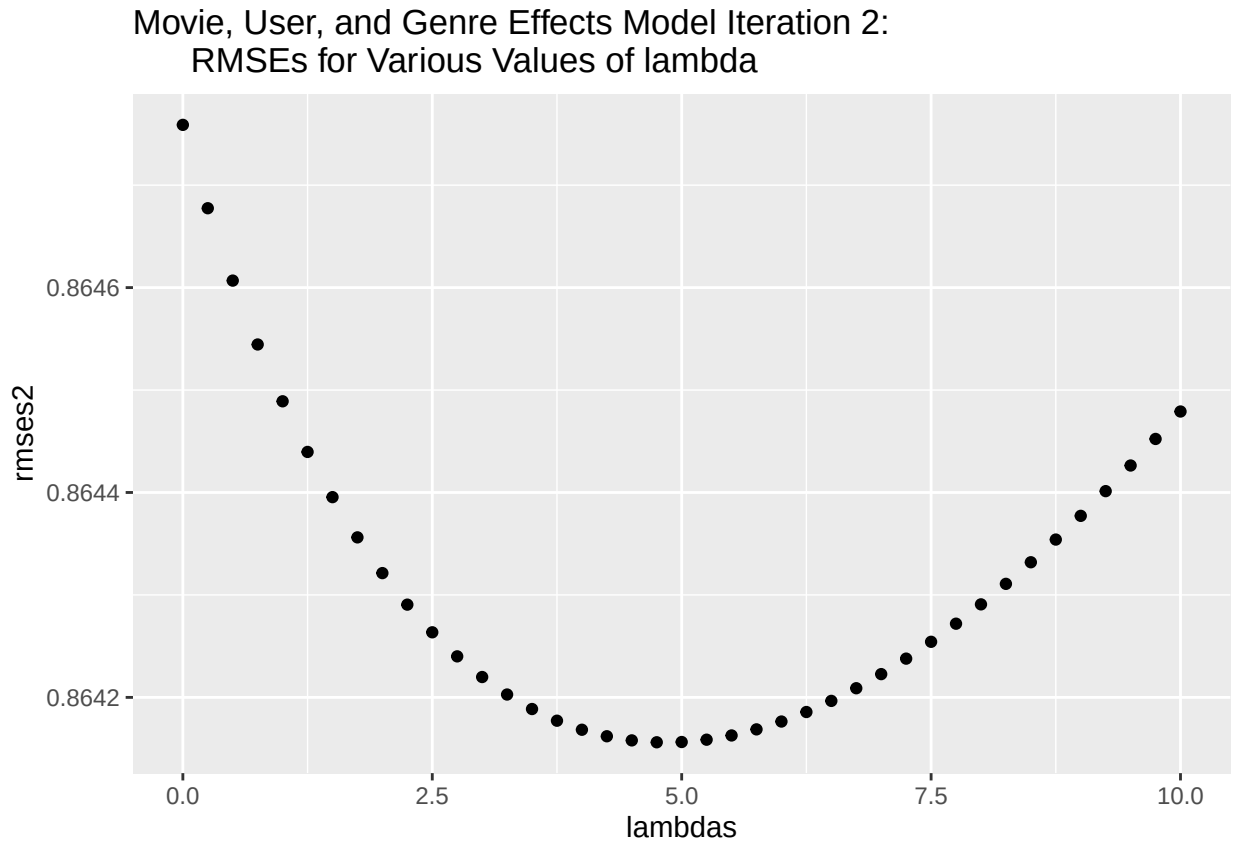



```
opt_lambda1 <- lambdas[which.min(rmses1)]
opt_lambda1
```

```
## [1] 4.75
```

```
########################## 2

rmses2 <- sapply(lambdas, function(lambda){
  b_i <- e2 %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu) / (n() + lambda))
  b_u <- e2 %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i) / (n() + lambda))
  b_g <- e2 %>% left_join(b_i, by = 'movieId') %>% left_join(b_u, by = 'userId') %>%
    group_by(genres) %>% summarize(b_g = sum(rating - mu - b_i - b_u) / (n() + lambda))

  predicted_ratings <- val2 %>% left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    left_join(b_g, by = 'genres') %>%
```

```
    mutate(pred = mu + b_i + b_u + b_g) %>% .$pred

  return(RMSE(val2$rating, predicted_ratings))
})

qplot(lambdas, rmses2,
      main = "Movie, User, and Genre Effects Model Iteration 2:
      RMSEs for Various Values of lambda")
```

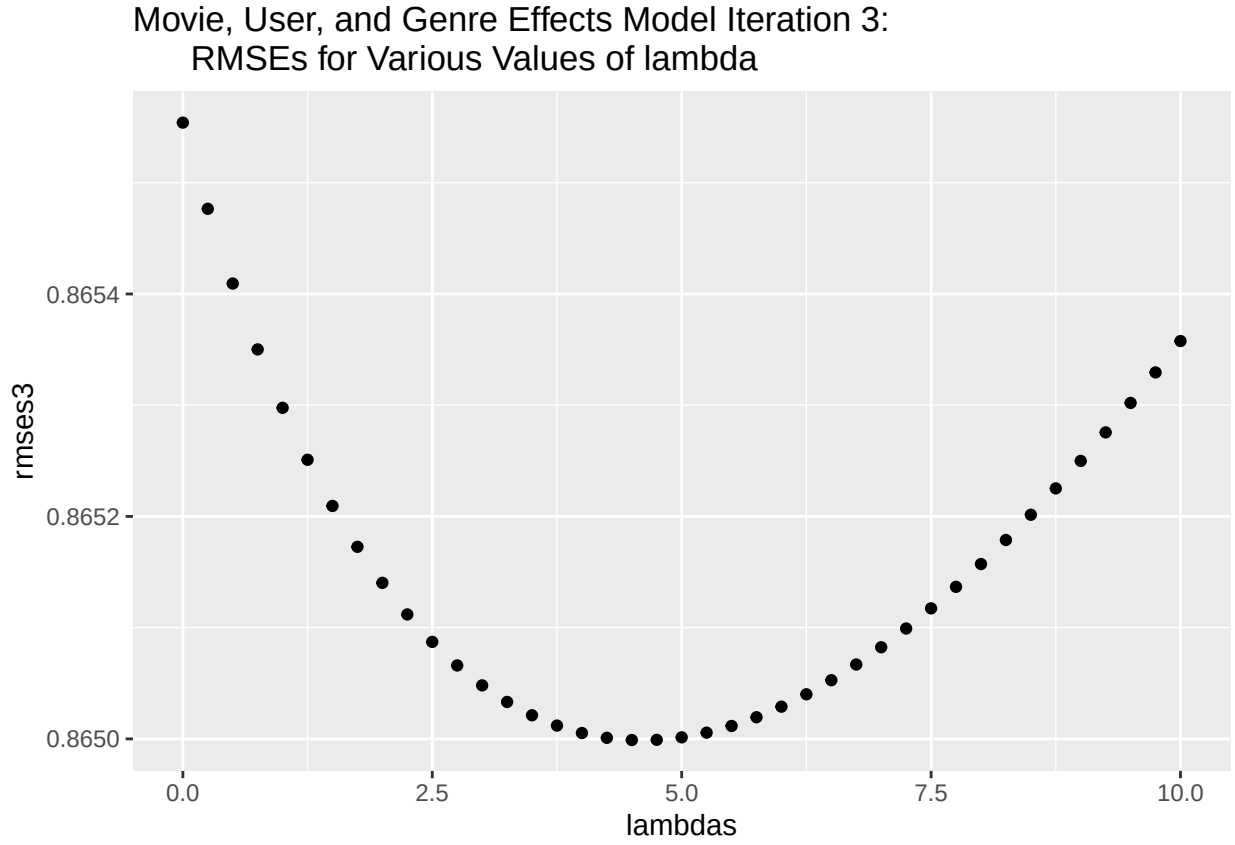



```
opt_lambda2 <- lambdas[which.min(rmses2)]
opt_lambda2
```

```
## [1] 4.75
```

```
########################## 3

rmses3 <- sapply(lambdas, function(lambda){
  b_i <- e3 %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu) / (n() + lambda))
  b_u <- e3 %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i) / (n() + lambda))
  b_g <- e3 %>% left_join(b_i, by = 'movieId') %>% left_join(b_u, by = 'userId') %>%
    group_by(genres) %>% summarize(b_g = sum(rating - mu - b_i - b_u) / (n() + lambda))

  predicted_ratings <- val3 %>% left_join(b_i, by = 'movieId') %>%
```

```
    left_join(b_u, by = 'userId') %>%
    left_join(b_g, by = 'genres') %>%
    mutate(pred = mu + b_i + b_u + b_g) %>% .$pred

  return(RMSE(val3$rating, predicted_ratings))
})

qplot(lambdas, rmses3,
      main = "Movie, User, and Genre Effects Model Iteration 3:
      RMSEs for Various Values of lambda")
```

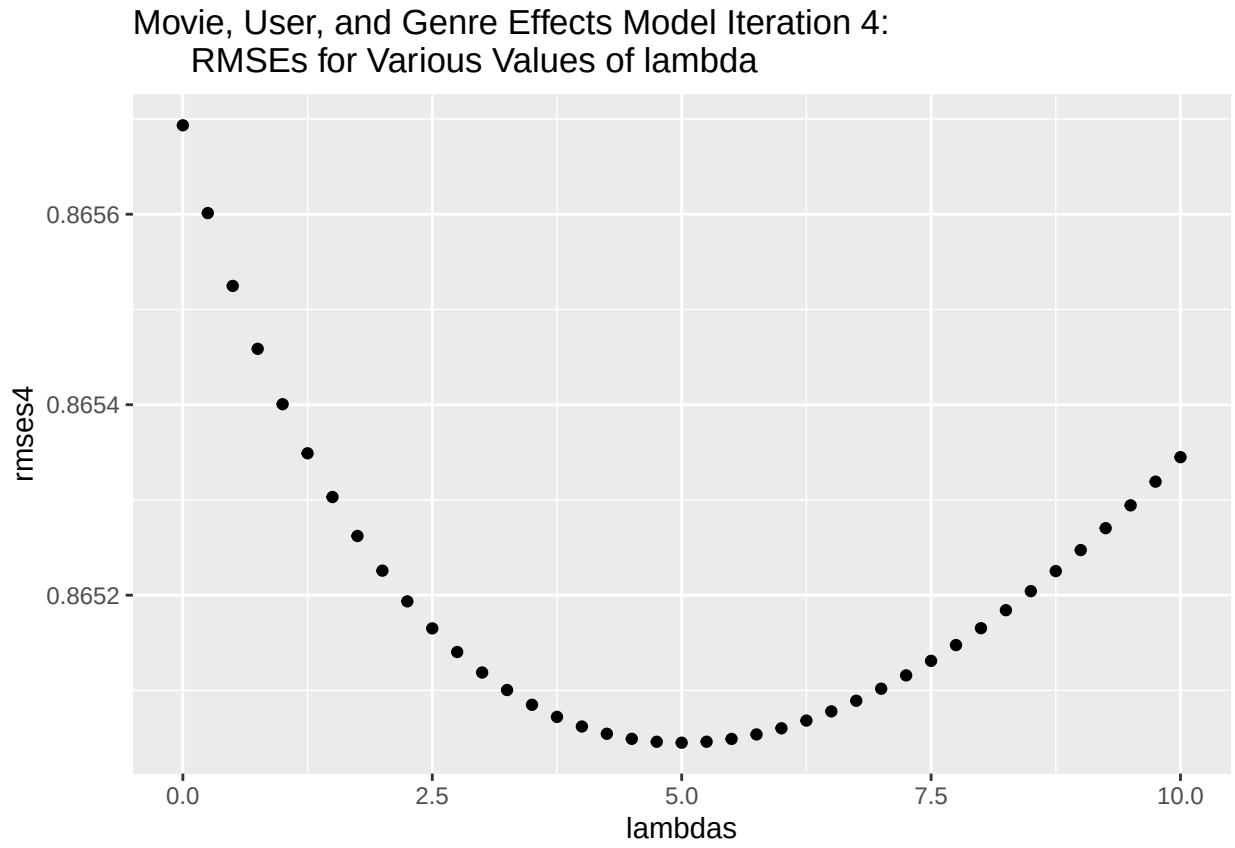



```
opt_lambda3 <- lambdas[which.min(rmses3)]
opt_lambda3
```

```
## [1] 4.5
```

```
########################## 4

rmses4 <- sapply(lambdas, function(lambda){
  b_i <- e4 %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu) / (n() + lambda))
  b_u <- e4 %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i) / (n() + lambda))
  b_g <- e4 %>% left_join(b_i, by = 'movieId') %>% left_join(b_u, by = 'userId') %>%
    group_by(genres) %>% summarize(b_g = sum(rating - mu - b_i - b_u) / (n() + lambda))
```

```
  predicted_ratings <- val4 %>% left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    left_join(b_g, by = 'genres') %>%
    mutate(pred = mu + b_i + b_u + b_g) %>% .$pred

  return(RMSE(val4$rating, predicted_ratings))
})

qplot(lambdas, rmses4,
      main = "Movie, User, and Genre Effects Model Iteration 4:
      RMSEs for Various Values of lambda")
```

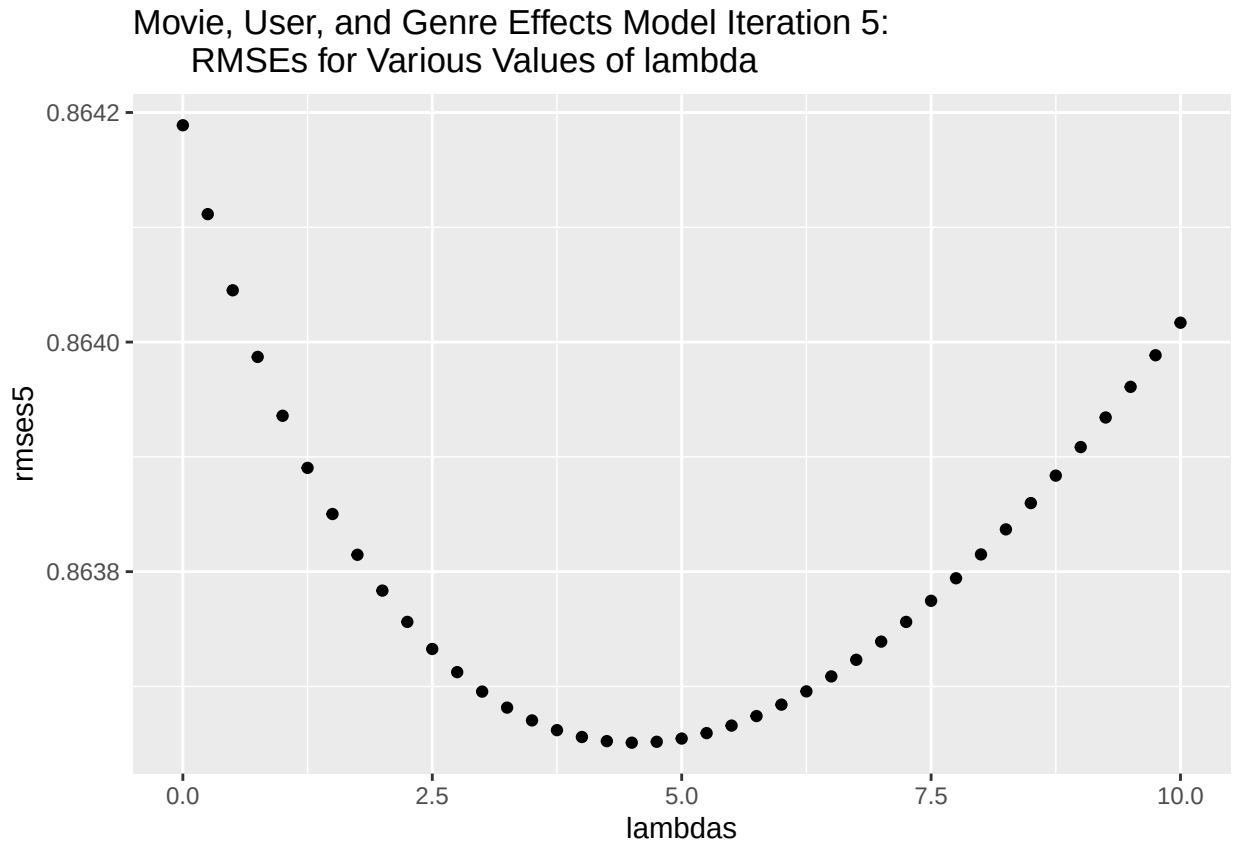



```
opt_lambda4 <- lambdas[which.min(rmses4)]
opt_lambda4
```

```
## [1] 5
```

```
########################## 5

rmses5 <- sapply(lambdas, function(lambda){
  b_i <- e5 %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu) / (n() + lambda))
  b_u <- e5 %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i) / (n() + lambda))
  b_g <- e5 %>% left_join(b_i, by = 'movieId') %>% left_join(b_u, by = 'userId') %>%
```

```
    group_by(genres) %>% summarize(b_g = sum(rating - mu - b_i - b_u) / (n() + lambda))

  predicted_ratings <- val5 %>% left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    left_join(b_g, by = 'genres') %>%
    mutate(pred = mu + b_i + b_u + b_g) %>% .$pred

  return(RMSE(val5$rating, predicted_ratings))
})

qplot(lambdas, rmses5,
      main = "Movie, User, and Genre Effects Model Iteration 5:
      RMSEs for Various Values of lambda")
```

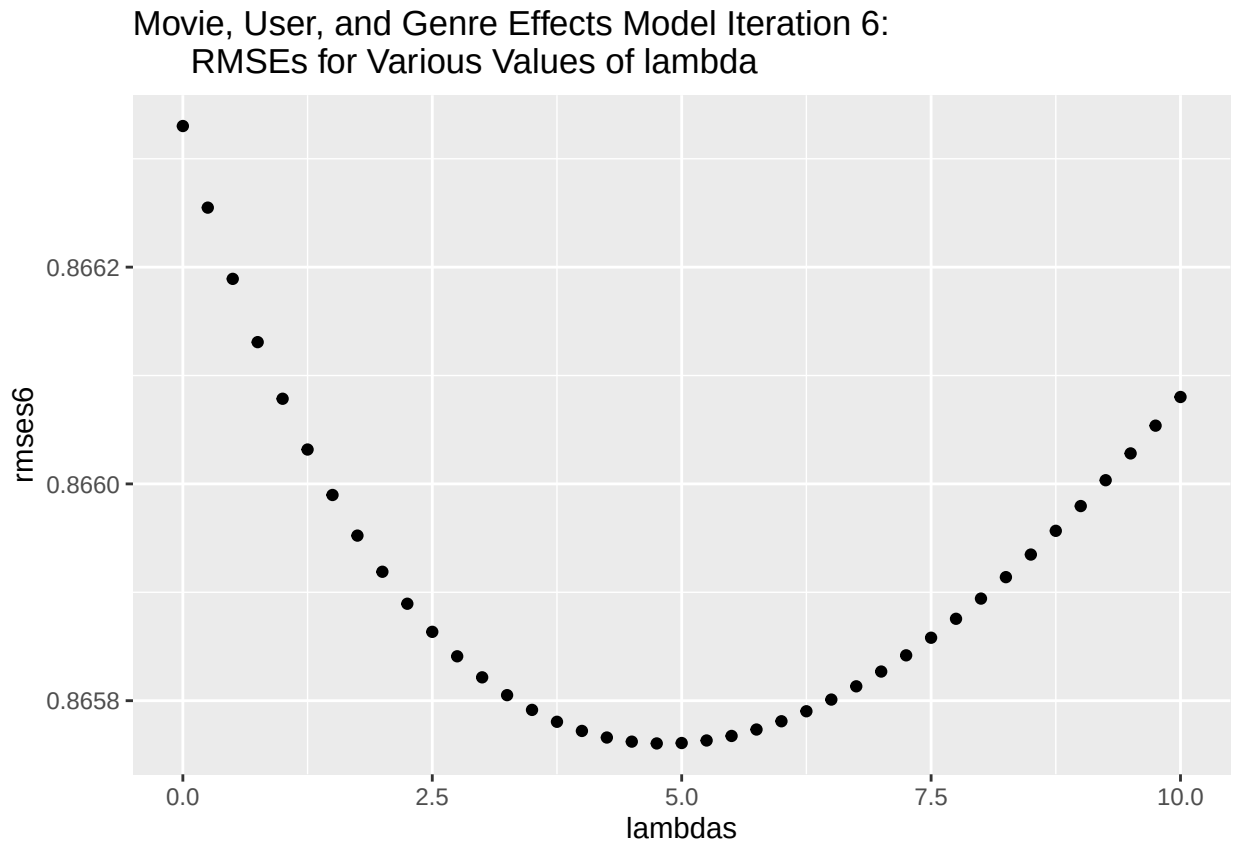



```
opt_lambda5 <- lambdas[which.min(rmses5)]
opt_lambda5
```

```
## [1] 4.5
```

```
########################## 6

rmses6 <- sapply(lambdas, function(lambda){
  b_i <- e6 %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu) / (n() + lambda))
  b_u <- e6 %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
```

```
    summarize(b_u = sum(rating - mu - b_i) / (n() + lambda))
  b_g <- e6 %>% left_join(b_i, by = 'movieId') %>% left_join(b_u, by = 'userId') %>%
    group_by(genres) %>% summarize(b_g = sum(rating - mu - b_i - b_u) / (n() + lambda))

  predicted_ratings <- val6 %>% left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    left_join(b_g, by = 'genres') %>%
    mutate(pred = mu + b_i + b_u + b_g) %>% .$pred

  return(RMSE(val6$rating, predicted_ratings))
})

qplot(lambdas, rmses6,
      main = "Movie, User, and Genre Effects Model Iteration 6:
      RMSEs for Various Values of lambda")
```

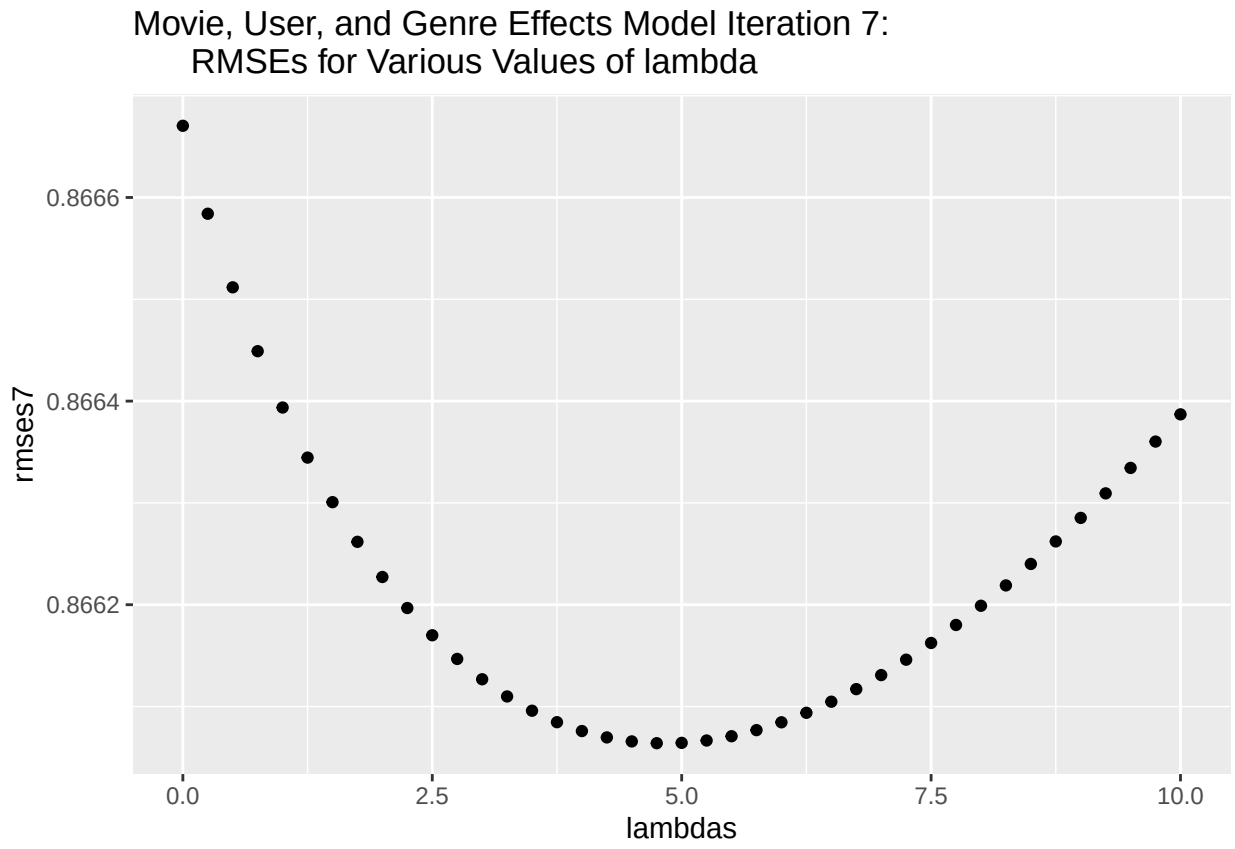



```
opt_lambda6 <- lambdas[which.min(rmses6)]
opt_lambda6
```

```
## [1] 4.75
```

```
########################## 7

rmses7 <- sapply(lambdas, function(lambda){
```

```
  b_i <- e7 %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu) / (n() + lambda))
  b_u <- e7 %>% left_join(b_i, by = "movieId") %>% group_by(userId) %>%
    summarize(b_u = sum(rating - mu - b_i) / (n() + lambda))
  b_g <- e7 %>% left_join(b_i, by = 'movieId') %>% left_join(b_u, by = 'userId') %>%
    group_by(genres) %>% summarize(b_g = sum(rating - mu - b_i - b_u) / (n() + lambda))

  predicted_ratings <- val7 %>% left_join(b_i, by = 'movieId') %>%
    left_join(b_u, by = 'userId') %>%
    left_join(b_g, by = 'genres') %>%
    mutate(pred = mu + b_i + b_u + b_g) %>% .$pred

  return(RMSE(val7$rating, predicted_ratings))
})

qplot(lambdas, rmses7,
      main = "Movie, User, and Genre Effects Model Iteration 7:
      RMSEs for Various Values of lambda")
```




```
opt_lambda7 <- lambdas[which.min(rmses7)]
opt_lambda7
```

```
## [1] 4.75
```

```
# Compute the value of optimal lambda using the results of our 7-fold cross-validation.

opt_lambda <- mean(c(opt_lambda1, opt_lambda2, opt_lambda3, opt_lambda4, opt_lambda5,
                     opt_lambda6, opt_lambda7))
opt_lambda
```

```
## [1] 4.714286
```

Results

Our efforts to regularize movie, user, and genre effects using K-fold cross-validation revealed **a value for optimal lambda of 4.71429**. This was the missing piece of our model. Armed with optimal lambda, our computations of movie, user, and genre effects could now include a penalty term that, in effect, gave less weight to movies, users, and genres with relatively few ratings.

Computing Regularized Movie, User, and Genre Effects

```
# Use optimal lambda to compute regularized movie, user, and genre effects.

lambda <- opt_lambda
mu <- mean(edx$rating)

b_i <- edx %>% group_by(movieId) %>% summarize(b_i = sum(rating - mu) / (n() + lambda))
b_u <- edx %>% left_join(b_i, by = 'movieId') %>% group_by(userId) %>%
  summarize(b_u = sum(rating - mu - b_i) / (n() + lambda))
b_g <- edx %>% left_join(b_i, by = 'movieId') %>% left_join(b_u, by = 'userId') %>%
  group_by(genres) %>% summarize(b_g = sum(rating - mu - b_i - b_u) / (n() + lambda))
```

The final form of our predictive model could now be represented as the following:

Prediction = 3.51237 + b_i (regularized movie effect for the particular movie) + b_u (regularized user effect for the particular user) + b_g (regularized genre effect for the particular genre)

Assessing the Accuracy of the Regularized Movie, User, and Genre Effects Model

All that was left to do was test. Using a new dataset (*validation*), we applied our algorithm to predict what ratings would be submitted by some of these same users for different movies than those in our initial *edx* training set. We then assessed the accuracy of these predictions by computing the RMSE.

```
# Use the Regularized Movie, User, and Genre Effects Model to predict ratings from
# the validation set.

predicted_ratings <- validation %>% left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = 'userId') %>%
  left_join(b_g, by = 'genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>% .$pred

# Assess the accuracy of the predictions by computing the RMSE of the Regularized Movie,
# User, and Genre Effects Model.
```

```
reg_movie_user_genre_rmse <- RMSE(validation$rating, predicted_ratings)
reg_movie_user_genre_rmse
```

```
## [1] 0.8634521
```

```
# Update our table of RMSE results

rmse_results <- bind_rows(rmse_results,
                data_frame(Method = "Regularized Movie, User, and Genre Effects Model",
                                     RMSE = reg_movie_user_genre_rmse))
knitr::kable(rmse_results[1:6, ], caption = "RMSE Results")
```

Table 13: RMSE Results

Method	RMSE
Random Model	1.0589588
Movie Effect Model	0.9426465
Movie + User Effects Model	0.8642948
Regularized Movie Effect Model	0.9425910
Regularized Movie and User Effects Model	0.8637400
Regularized Movie, User, and Genre Effects Model	0.8634521

As the table of RMSE results shows, including movie, user, and genre effects, and regularizing each of them in our predictive model allowed us to clearly achieve the project's goal: an RMSE at or below 0.8649.

Conclusion

With a **final RMSE of 0.8634521**, our ratings prediction model is sufficiently accurate to be incorporated into a movie recommendation system.

This is not to say that the model can't be further refined to improve its accuracy. Other effects could have been considered, such as the year the movie was released, or the length of time between movie release and rating submission. They may have revealed further structure in the ratings data and thus enhanced the accuracy of the predictive model.

Even seemingly random effects such as the day of the week of the rating submission, or the hour of the day of the rating submission, may have revealed patterns in the data useful to our model.

It should also be noted that computer systems with greater system resources (memory, cpu, etc.) would have allowed the use of more sophisticated regression and cross-validation methods that may have produced superior results.