Report on Movie Recommendations

Abigail Motley

Overview

The goal of this report is to generate a machine learning model to recommend movies to users by predicting the rating a user would give a particular movie. We will use the 10M version of the `MovieLens` dataset to build and validate the model. First, we will clean and split the data into our training and final validation sets. Next, we'll analyze the data and use that insight to build and test intermediary models. Finally, we'll choose the best model to run our validation set on. Our final model is linear and has an RMSE of 0.8644514 on the validation set.

Analysis

Preparing the Data

The `MovieLens` dataset from `http://files.grouplens.org/datasets/movielens/ml-10m.zip` is split into two separate .dat files called `ratings` and `movies`. We need to download the data from both files, convert each into a dataframe in R, add column names (Movie, User, Genre, Timestamp, and Rating), convert the data into the appropriate type (character, numeric, etc), and finally join them together into a single dataframe.

Next, we need to split our data into our test and final validation sets called `edx` and `validation`. The `edx` set will be used to explore our data and create intermediary models. The `validation` set will only be used at the end to test our best model. In this case, we create a partition where 10% of the data is used for the `validation` set. We are careful to ensure that the userIds and movieIds that appear in the `validation` set are also in the `edx` set.

We now have `edx` and `validation` sets:

```
dim(edx)
```

```
## [1] 9000055       6
```

```
dim(validation)
```

```
## [1] 999999      6
```

Finally, we want to split the `edx` dataset into two separate `train` and `test` subsets that can be used for intermediate RMSE validation. We use 10% of the `edx` data for the `test` set, and again ensure that the userIds and movieIds that appear in the `test` set are also in the `train` set.

```
dim(train)
```

```
## [1] 8100065       6
```

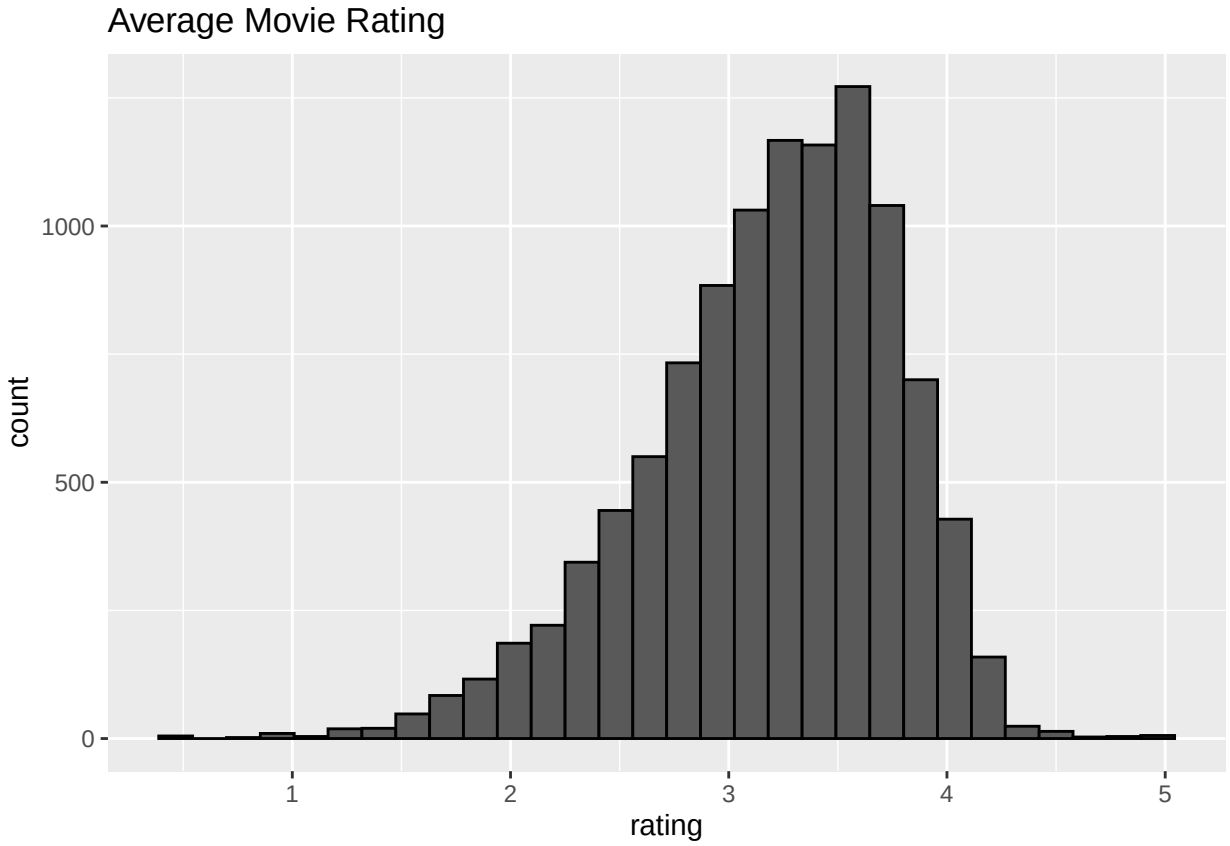
```
dim(test)
```

```
## [1] 899990      6
```

Now that we've prepared our data, we will explore the properties of various predictors and decide how to use them to create our prediction model. Potential preditors are: Movie, User, Genre, and Timestamp.

Movie Effect

To see if the movie may be an interesting predictor for our model, let's plot the average movie ratings.



Since the results have some variety, it seems like movies will be a useful predictor to add to our model.

We can use a linear model which considers the movie in the prediction: `Y_u,i = mu + b_i + epsilon_u,i`. `mu` is the true rating for all movies, which we can estimate using the mean. `b_i` is the movie effect, or the average ranking for movie i. `epsilon` represents the error. Since running R's `lm` function will be too slow due to the size of the dataset, we can instead estimate `b_i` by using `Y_u,i - mu` for each movie i.

```
mu <- mean(train$rating)
b_i_estimates <- train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

Let's create the model using the b_i_estimates.

```
movie_effect_model <- mu + test %>%
  left_join(b_i_estimates, by='movieId') %>%
  pull(b_i)
```

Find the RMSE of the model using the `test` set. Note we are not using the `validation` set since this will only be used to measure the performance of our final model.

```
rmse_movie <- RMSE(movie_effect_model, test$rating)
rmse_movie
```

```
## [1] 0.9429615
```

Let's explore what our model is estimating and see if we can do better. Let's look at the top 10 and bottom 10 rated movies.

```
top_10_movies
```

```
##  [1] "Hellhounds on My Trail (1999)"
##  [2] "Satan's Tango (Sátántangó) (1994)"
##  [3] "Shadows of Forgotten Ancestors (1964)"
##  [4] "Fighting Elegy (Kenka erejii) (1966)"
##  [5] "Sun Alley (Sonnenallee) (1999)"
##  [6] "Blue Light, The (Das Blaue Licht) (1932)"
##  [7] "Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)"
##  [8] "Life of Oharu, The (Saikaku ichidai onna) (1952)"
##  [9] "Human Condition II, The (Ningen no joken II) (1959)"
## [10] "Human Condition III, The (Ningen no joken III) (1961)"
```

```
bottom_10_movies
```

```
##  [1] "Besotted (2001)"
##  [2] "Hi-Line, The (1999)"
##  [3] "Accused (Anklaget) (2005)"
##  [4] "Confessions of a Superhero (2007)"
##  [5] "War of the Worlds 2: The Next Wave (2008)"
##  [6] "SuperBabies: Baby Geniuses 2 (2004)"
##  [7] "Disaster Movie (2008)"
##  [8] "From Justin to Kelly (2003)"
##  [9] "Hip Hop Witch, Da (2000)"
## [10] "Criminals (1996)"
```

We notice our model's top 10 best and worst rated movies are not very well-known. Let's see how many ratings each of these movies have.

```
top_10_movie_count_rated
```

```
##  [1] 1 1 1 1 1 1 4 2 4 4
```

```
bottom_10_movie_count_rated
```

```
##  [1]   1   1   1   1   2  47  30 183  11   1
```

It looks like most of the movies in the top and bottom 10 don't have a lot of users who rated them.

In general, we can also see that a lot of movies have very few ratings.

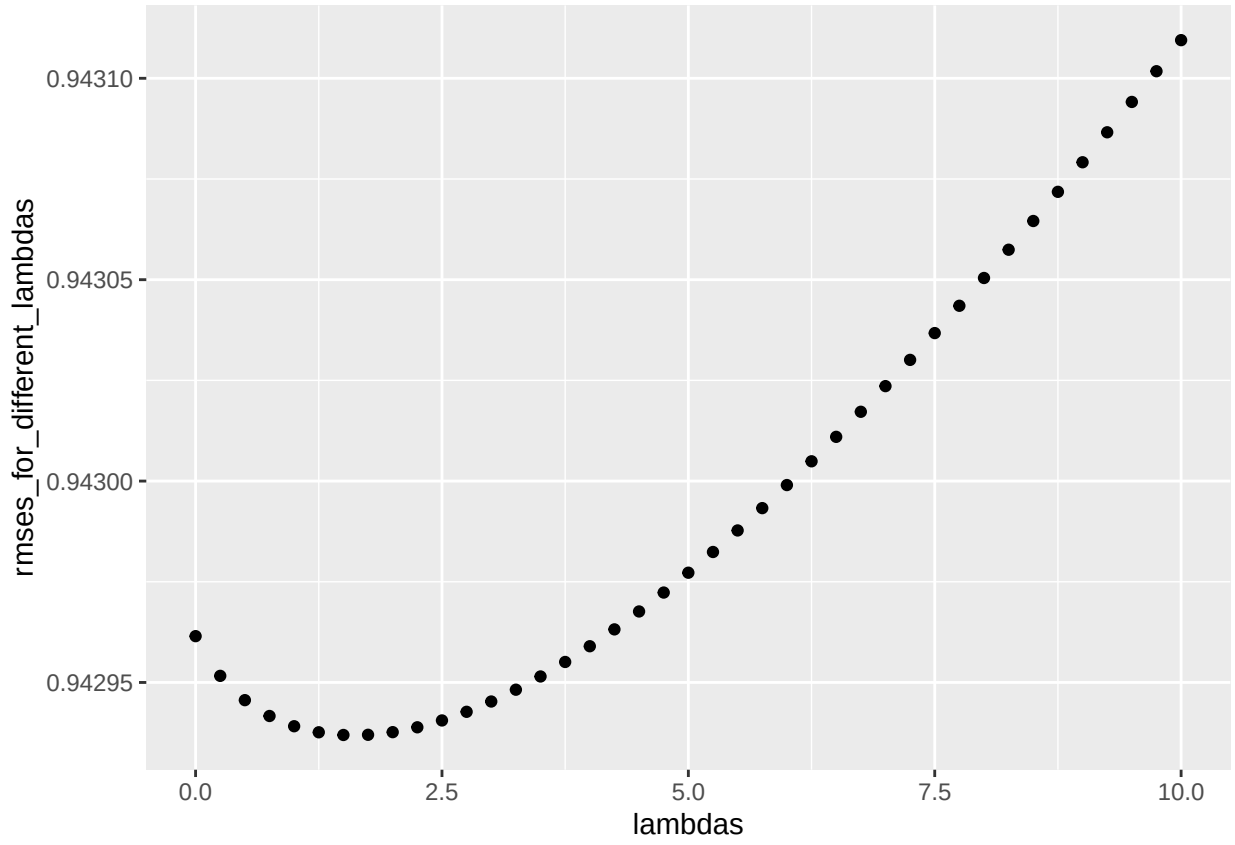
```
edx %>%
  group_by(movieId) %>%
  summarize(numRatings=n()) %>%
  filter(numRatings < 25) %>%
  summarize(moviesWithLessThan25Ratings = n())
```

```
## # A tibble: 1 x 1
##   moviesWithLessThan25Ratings
##                         <int>
## 1                        2228
```

Since we know predictions made with a low number of data points tend to be less accurate, we should give these estimates less weight in our model. To reduce the effect of these predictions made with small data points, we can use regularization. We will need to add a penalty lambda to our estimate of b_i: `b_i_regularized = sum(ratings - mu)/(lambda + count_ratings)`.

To choose the right value of lambda, we can use cross valdation.

```
lambdas <- seq(0, 10, 0.25)
rmses_for_different_lambdas <- sapply(lambdas, function(l){
  mu <- mean(train$rating)
  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating-mu)/(n()+l))
  movie_effect_model_reg <- test %>%
    left_join(b_i, by='movieId') %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(movie_effect_model_reg, test$rating))
})
qplot(lambdas, rmses_for_different_lambdas)
```



```
best_lambda <- lambdas[which.min(rmses_for_different_lambdas)]
best_lambda
```

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

Let's create the model using the regularized b_i_estimates and the best lambda.

```
b_i_estimates_reg <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+best_lambda), n_i = n())
movie_effect_model_reg <- test %>%
  left_join(b_i_estimates_reg, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)
```

The RMSE of this model is 0.942937, which is slightly better than before.

```
rmse_movie
```

```
## [1] 0.9429615
```

Let's see if our top 10 and bottom 10 movies makes more sense after regularization.

```
top_10_movies_reg
```

```
##  [1] "Shawshank Redemption, The (1994)"
##  [2] "More (1998)"
##  [3] "Godfather, The (1972)"
##  [4] "Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)"
##  [5] "Human Condition II, The (Ningen no joken II) (1959)"
##  [6] "Human Condition III, The (Ningen no joken III) (1961)"
##  [7] "Usual Suspects, The (1995)"
##  [8] "Schindler's List (1993)"
##  [9] "Rear Window (1954)"
## [10] "Casablanca (1942)"
```

```
bottom_10_movies_reg
```

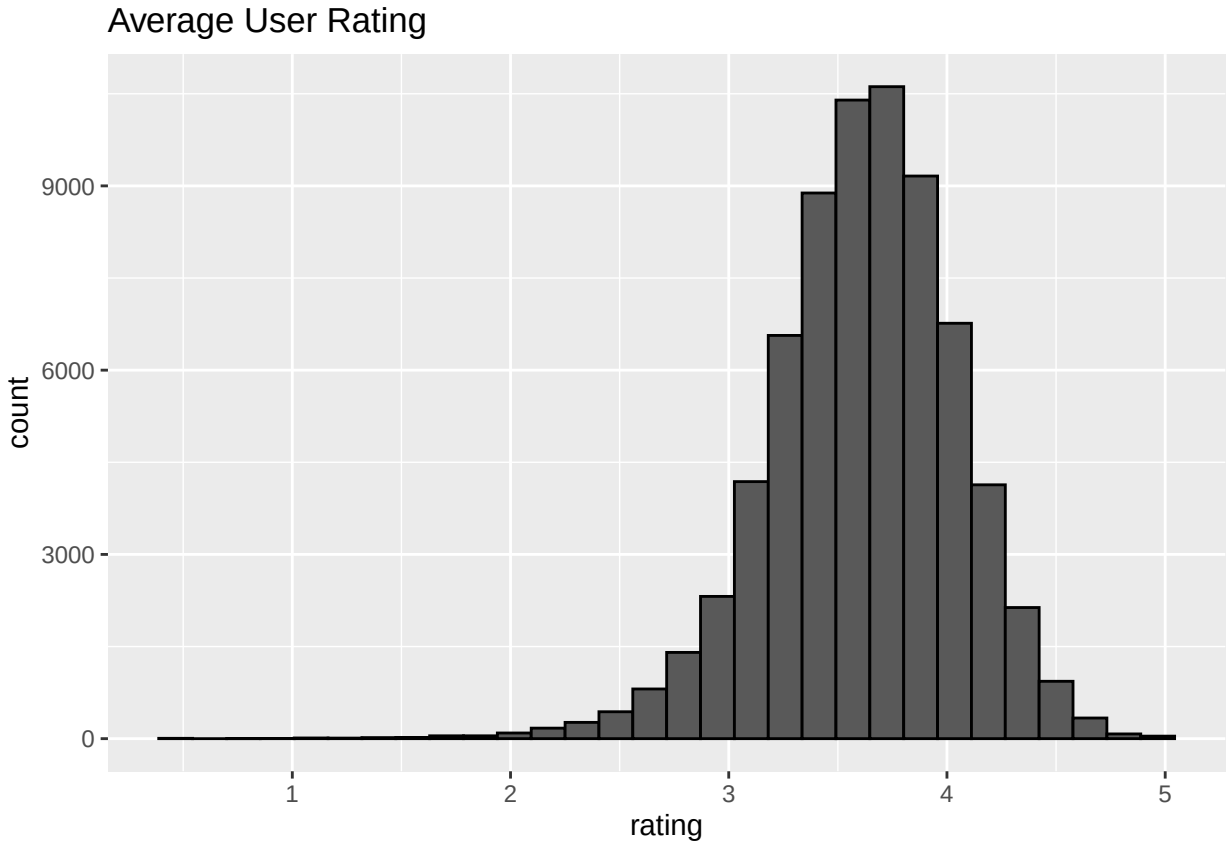
```
##  [1] "SuperBabies: Baby Geniuses 2 (2004)"
##  [2] "From Justin to Kelly (2003)"
##  [3] "Disaster Movie (2008)"
##  [4] "Pokémon Heroes (2003)"
##  [5] "Barney's Great Adventure (1998)"
##  [6] "Carnosaur 3: Primal Species (1996)"
##  [7] "Glitter (2001)"
##  [8] "Gigli (2003)"
##  [9] "Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002)"
## [10] "Hip Hop Witch, Da (2000)"
```

These movies make a lot more sense.

Next, let's explore the users.

User Effect

To see if the user may be an interesting preditor for our model, let's plot the average user ratings.



Since the results have some variety, it seems like user will be a useful predictor to add to our model.

Our modified model is: `Y_u,i = mu + b_i + b_u + epsilon_u,i`. `b_u` is the user effect, or the average ranking for user u. This is estimated by `Y_u,i - mu - b_i` for each user u. Remember that last time we had to use regularization because there were movie that didn't have many ratings. Let's see if we need to regularize users as well.

```
edx %>%
  group_by(userId) %>%
  summarize(numRatings=n()) %>%
  filter(numRatings < 25) %>%
  summarize(usersWithLessThan25Ratings = n())
```

```
## # A tibble: 1 x 1
##   usersWithLessThan25Ratings
##                        <int>
## 1                      10042
```

We can see that several users hardly rated any movies, so we'll need to use regularization again: `b_u_regularized = sum(ratings - mu - b_i)/(lambda + count_ratings)`

Let's create the model using the regularized b_i_estimates, b_u_estimates, and the best lambda.

```
b_i_estimates_reg <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+best_lambda), n_i = n())
b_u_estimates_reg <- train %>%
  left_join(b_i_estimates_reg, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u=sum(rating-b_i-mu)/(n()+best_lambda))
movie_user_effect_model_reg <- test %>%
  left_join(b_i_estimates_reg, by='movieId') %>%
  left_join(b_u_estimates_reg, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
```

The RMSE of this model is 0.8641362, which is a big improvement.

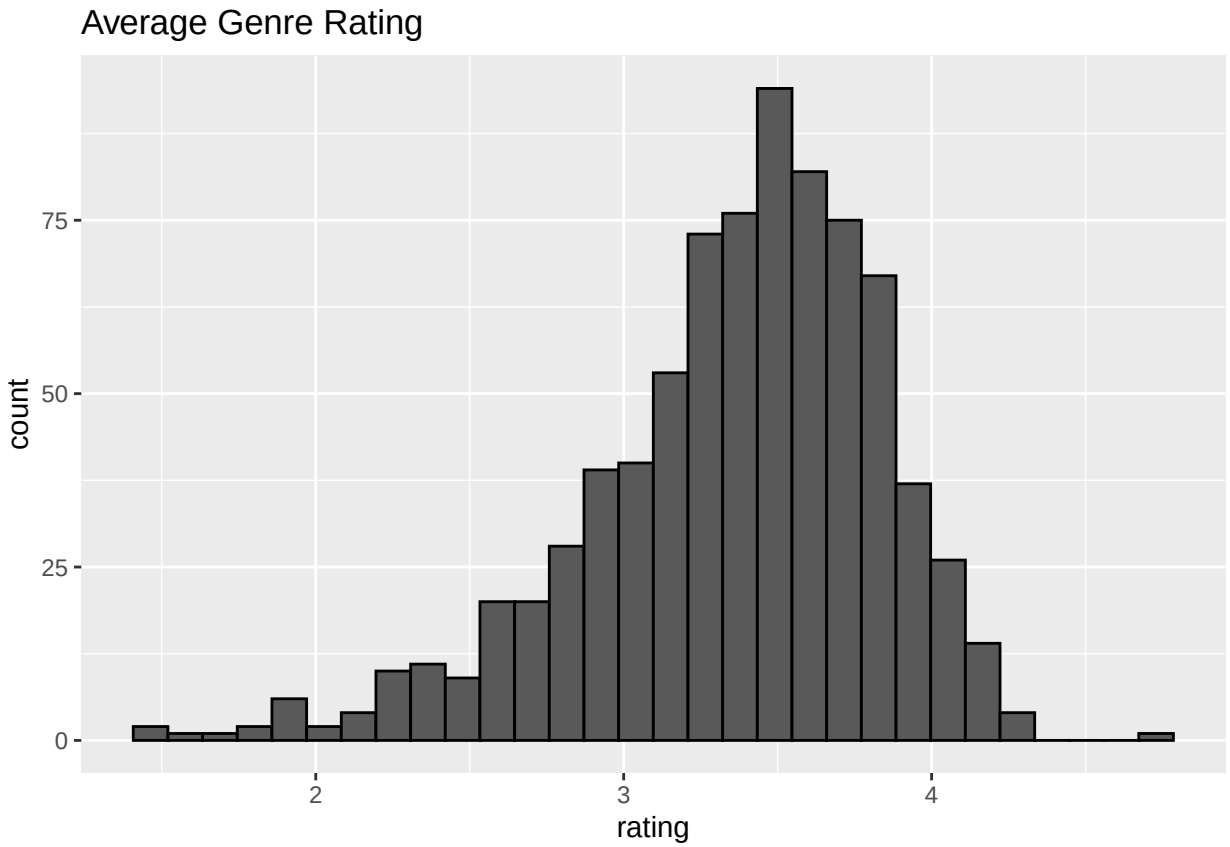
```
rmse_movie_user_reg
```

```
## [1] 0.8641362
```

Next, let's explore the genres.

Genre Effect

To see if the genre may be an interesting preditor for our model, let's plot the average genre ratings.



Since the results have some variety, it seems like genre will be a useful predictor to add to our model.

Our modified model is: `Y_u,i = mu + b_i + b_u + b_g epsilon_u,i.` `b_g` is the genre effect, or the average ranking for genre g. It can be estimated by `Y_u,i - mu - b_i - b_u` for each genre g. Let's see if we need to regularize genres.

```
edx %>%
  group_by(genres) %>%
  summarize(numRatings=n()) %>%
  filter(numRatings < 25) %>%
  summarize(genresWithLessThan25Ratings = n())
```

```
## # A tibble: 1 x 1
##   genresWithLessThan25Ratings
##                         <int>
## 1                          67
```

We can see that some genres don't have many ratings, so let's regularize: `b_g_regularized = sum(ratings - mu - b_i - b_u)/(lambda + count_ratings)`

Let's create the model using the regularized b_i_estimates, b_u_estimates, and the best lambda (4.75).

```
b_i_estimates_reg <- train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+best_lambda), n_i = n())
b_u_estimates_reg <- train %>%
  left_join(b_i_estimates_reg, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u=sum(rating-b_i-mu)/(n()+best_lambda))
b_g_estimates_reg <- train %>%
  left_join(b_i_estimates_reg, by='movieId') %>%
  left_join(b_u_estimates_reg, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g=sum(rating-b_i-b_u-mu)/(n()+best_lambda))
movie_user_genre_effect_model_reg <- test %>%
  left_join(b_i_estimates_reg, by='movieId') %>%
  left_join(b_u_estimates_reg, by='userId') %>%
  left_join(b_g_estimates_reg, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
```

The RMSE of this model is 0.8638141, which is a slight improvement.

```
rmse_movie_user_genres_reg
```

```
## [1] 0.8638141
```

We have some pretty good models, so let's move on to the results.

Results

Here are the intermediate RMSEs of the models:

```
## # A tibble: 4 x 2
##   method                                     RMSE
##   <chr>                                     <dbl>
## 1 Movie Effect Model                        0.943
## 2 Regularized Movie Effect Model            0.943
## 3 Regularized Movie + User Effect Model     0.864
## 4 Regularized Movie + User + Genre Effect Model 0.864
```

Let's Choose the best model, "Regularized Movie + User + Genre Effect Model", run it on validation set to find our final RMSE.

```
b_i_estimates_reg <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+best_lambda), n_i = n())
b_u_estimates_reg <- edx %>%
  left_join(b_i_estimates_reg, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u=sum(rating-b_i-mu)/(n()+best_lambda))
b_g_estimates_reg <- edx %>%
  left_join(b_i_estimates_reg, by="movieId") %>%
  left_join(b_u_estimates_reg, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g=sum(rating-b_i-b_u-mu)/(n()+best_lambda))
movie_user_genre_effect_model_reg <- validation %>%
  left_join(b_i_estimates_reg, by='movieId') %>%
  left_join(b_u_estimates_reg, by='userId') %>%
  left_join(b_g_estimates_reg, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)
final_rmse <- RMSE(movie_user_genre_effect_model_reg, validation$rating)
final_rmse
```

```
## [1] 0.8646574
```

The final RMSE of the validation set is 0.8644514.

Conclusion

To summarize, we've shown how to build a movie recommendation system using the `MovieLens` dataset. This process included data cleaning, data analysis to guide the model building, and evaluating and selecting the best model. In the end, we were able to build a linear model that performs with a RMSE of 0.8644514.