MovieLens Recommendation System

Report for EdX Data Science Capstone

Evan Martin

December, 2020

Introduction

Recommendation systems are often used to help users of a product better choose their selections. This can be applied to books, food and many other products and services. It is also an excellent opportunity to apply statistical methods and train data science algorithms. We are going to be looking at a system for movies.

Using a truncated version of the MovieLens dataset, this document details the steps of creating a movie recommendation system. We will first explore the data to help us understand the structure, and then analyze the available information to gain insight. From there we will go through the steps taken to construct our prediction system and then improve upon it. At the end, a summary of the method and results will be provided.

Initial Setup

```
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")

library(tidyverse)
library(caret)
library(data.table)

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(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))

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

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `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 userId and movieId in validation set are also in edx set
validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# 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)

set.seed(1, sample.kind = "Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

small_train_set <- edx[-test_index,]
small_test_set <- edx[test_index,]

RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2, na.rm = T))
}
```

Method & Analysis

The dataset used, MovieLense, truncated and titled 'edx' in the accompanying r script, has 9000055 rows, each row a review, with 6 columns: userId, movieId, rating (0.5-5), timestamp, title, and genres, the latter a combination of all applicable genres for that movie. There are 10677 unique movies in the dataset and 69878 unique users.

In order to make a movie recommendation system we start with the knowledge that not all users have reviewed all movies and therefore we can make predictions of what a given users rating would be once they do watch a given movie. The baseline (and most naive) estimation we can use is the arithmetic average for all movies, 3.5, or the most given rating, 4.

```
mean(edx$rating)
```

```
## [1] 3.512465
```

```
median(edx$rating)
```

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

One step beyond this we can take into account the genre of a movie and use the average for that given genre. And as a movie can have multiple genres attributed to it, we could again take the average of those averages. For instance, all other variables being held constant, the Romance genre has an average rating of 3.55

```
edx %>% filter(str_detect(genres, 'Romance')) %>% select(rating) %>% colMeans()
```

```
##   rating
## 3.553813
```

However if we look at 'pure' Romance, films that have only that single genre, we see the average is lower, at 3.26, signifying that in this situation, on average other genres give movies a ratings boost.

```
edx %>% filter(str_detect(genres, '^Romance$')) %>% select(rating) %>% colMeans()
```

```
##   rating
## 3.266528
```

Conversely, we can see the opposite with Drama. Movies that are soley Drama tend to have higher ratings (3.71) than movies that are Drama plus other genres (3.67). A film maker might conclude then to always throw in some Drama in order to boost ratings. In terms of a recommendation systems, we might be able to leverage this knowledge later.

```
edx %>% filter(str_detect(genres, '^Drama$')) %>% select(rating) %>% colMeans()
```

```
##   rating
## 3.712364
```

```
edx %>% filter(str_detect(genres, 'Drama')) %>% select(rating) %>% colMeans()
```

```
##   rating
## 3.673131
```

Now that we have a bit of an understanding of the dataset we are working with, let's get back to our recommendation system. Before we get into it, first we need to define how we are actually going to judge the accuracy of our estimation system, and if we have improved upon the naive approach of using the mean. As we take measurements, we will continue to explore our dataset to further refine our prediction.

In order to rate our system, we will split the 'edx' dataset into a test set containing 20% of the data and a mutually exclusive training set containing the other 80%. From here we can 'train' and 'test' our model of just using the average to set a baseline off which we can try to improve. Using a Root Square Mean Error (RMSE) as a general purpose metric we can measure how far our predicted values, generated using the training set, are from the observed values.

```
mu <- mean(train_set$rating)
sqrt(mean((test_set$rating - mu)^2))
```

```
## [1] 1.059909
```

After calculating the average rating within the training set, we can subtract it from the ratings of the test set to generate our errors. We then square the errors, forcing them all to be positive, take the average and transform back by taking the square root. We get an RMSE of 1.06. Not great.

NOTE: *Due to technical constraints, we will be relying on functions that are closer to pure math in order to accomodate RStudio memory limitations. Our baseline algorithm is the formula for the mean.* sum(rating)/length(rating)

We want to get to a point where we have an algorithm that has accurate estimates of the effects (or betas) of all relevant predictors. Since the baseline method is lumping all movies together, it seems we could make a lot better estimation by calculating the mean of each movie and finding the marginal effect of that individual movie mean, independent from the overall mean. We will create a variable movie_avgs which contains 'b_m', standing for the beta of movies.

```
movie_avgs <- train_set %>%
    group_by(movieId) %>%
    summarize(b_m = mean(rating - mu))
```

And generate predictions using both the overall mean and the effect of individual movie means.

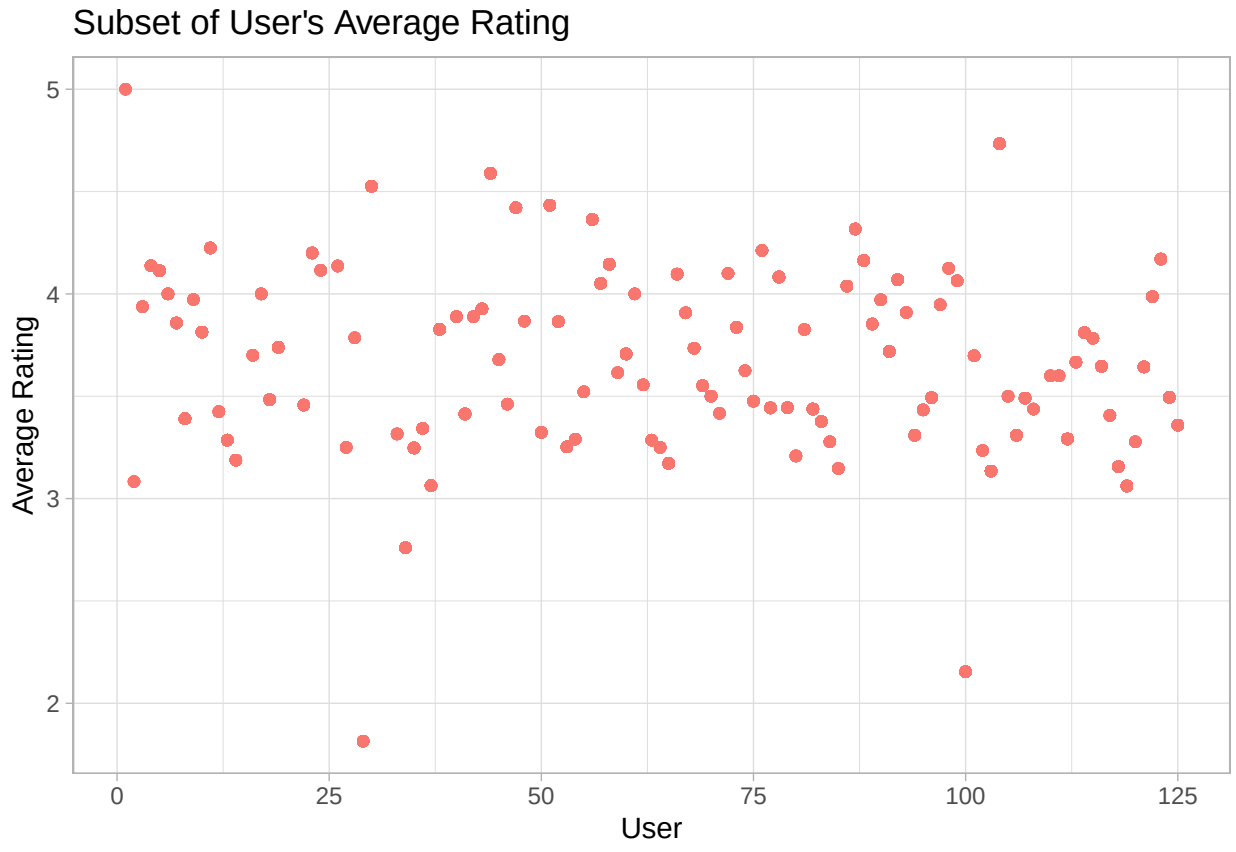
```
predicted_ratings <- test_set %>%
    left_join(movie_avgs, by='movieId') %>%
    summarize(pred = mu + b_m) %>%
    .$pred

RMSE(test_set$rating, predicted_ratings)
```

```
## [1] 0.9437429
```

This gives us a slightly more accurate prediction by reducing the RMSE to 0.943

The next logical step it would seem is to look at users in the same fashion. Perhaps there are users who are harsh critics and others that love everything they see.



Generate user averages, isolating the effect from the overall average and the individual movie average.

```
user_avgs <- train_set %>%
     left_join(movie_avgs, by='movieId') %>%
     group_by(userId) %>%
     summarize(b_u = mean(rating - mu - b_m))
```

And re-generate our predictions.

```
predicted_ratings <- test_set %>%
     left_join(movie_avgs, by='movieId') %>%
     left_join(user_avgs, by='userId') %>%
     mutate(pred = mu + b_m + b_u) %>%
     .$pred

RMSE(test_set$rating, predicted_ratings)
```

```
## [1] 0.865932
```

Indeed, our RSME is reduced even further to 0.8659

Next, let us assume there have been trends across time, periods that produce more higher rated movies than others.

NOTE: *The timestamp column indicates when the entry was added to the dataset, so we need to rely on the year being present in the title. From the str_sub documenation we can parse backwards from the end of the title to obtain the 5th thru 2nd characters from the end. This way we can avoid matching other parantheses or four digit numbers in titles.*

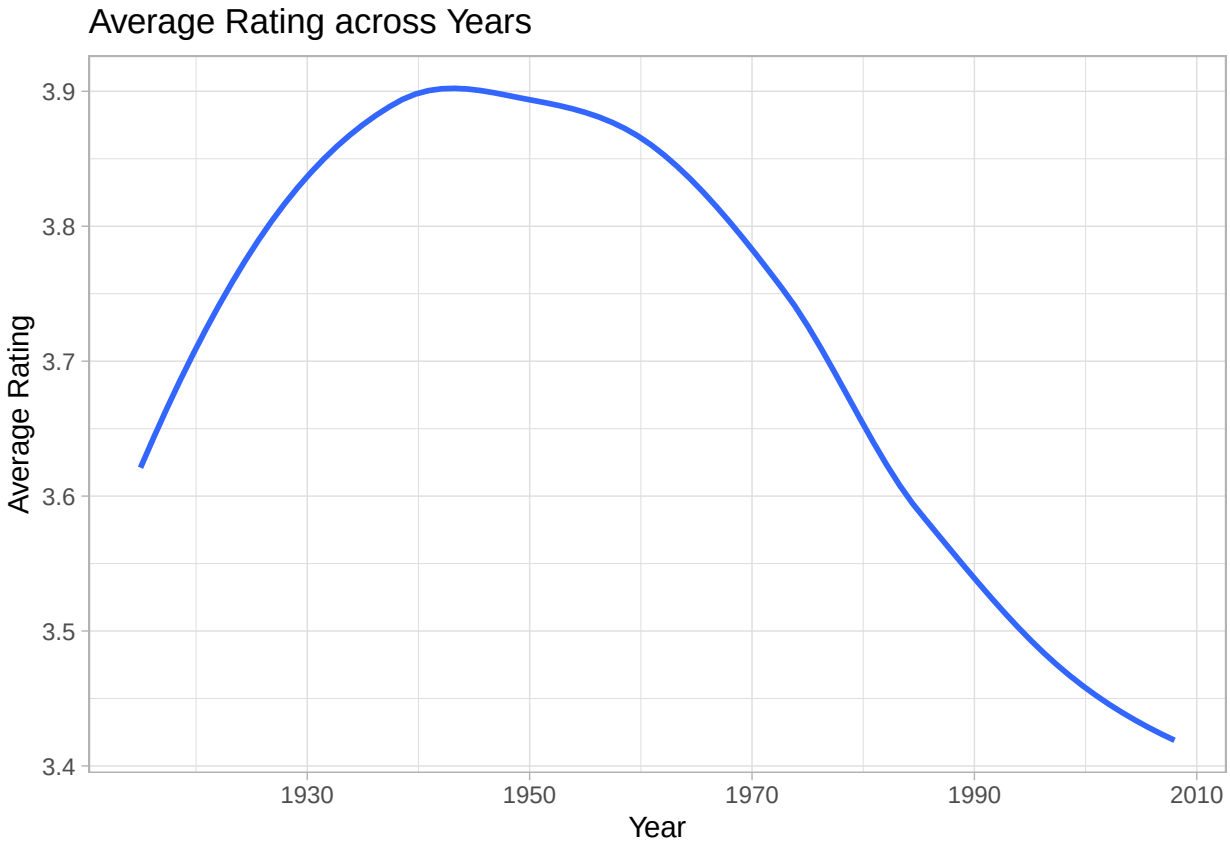
```
edx[1]$title
```

```
## [1] "Boomerang (1992)"
```

```
str_sub(edx[1]$title, -5, -2)
```

```
## [1] "1992"
```

Looking at yearly averages, we see there were some relatively high ratings in 1940s that have been trending down since.



Let's see how much it helps our model by isolating the effect from our other betas.

```
year_avgs <- train_set %>%
    mutate(year = as.numeric(str_sub(title,-5,-2))) %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    group_by(year) %>%
    summarize(b_y = mean(rating - mu - b_m - b_u))
```

Predictions with this new information.

```
predicted_ratings <- test_set %>%
    mutate(year = as.numeric(str_sub(title,-5,-2))) %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    left_join(year_avgs, by='year') %>%
    mutate(pred = mu + b_m + b_u + b_y) %>%
    .$pred

RMSE(test_set$rating, predicted_ratings)
```

```
## [1] 0.8656117
```

Our RMSE, nearly unchanged at 0.8656

The last variable we have to squeeze any help out of is genre. For this we will leave the genres grouped as they are. (For now, leaving out the year-average-beta since it had a small effect)

```
genre_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_m - b_u))

predicted_ratings <- test_set %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    left_join(genre_avgs, by='genres') %>%
    mutate(pred = mu + b_m + b_u + b_g) %>%
    .$pred

RMSE(test_set$rating, predicted_ratings)
```

```
## [1] 0.8655941
```

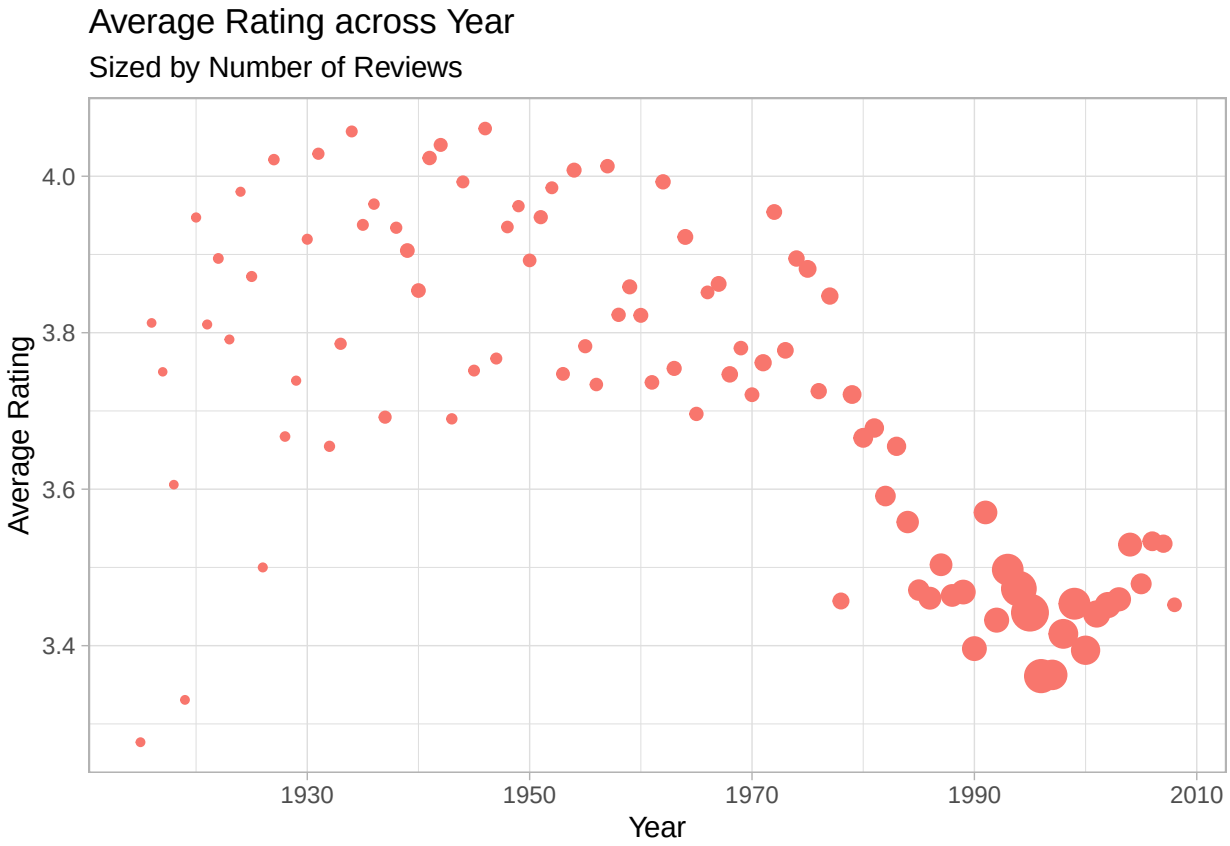
RMSE: 0.8655

We have looked at all variables, but we have overlooked one thing while making our calculations. Let's look again at some of the data. With a little careful inspection we can see that we need to be careful with our approach. For instance, if we inspect movies that have an average rating of 5, we'll see there are only a handful, and they all have only a few ratings.

```
train_set %>% group_by(movieId) %>%
  summarize(avg = mean(rating), n = length(rating)) %>%
  filter(avg==5)
```

```
## # A tibble: 9 x 3
##   movieId   avg     n
##     <dbl> <dbl> <int>
## 1    3226     5     1
## 2    5194     5     3
## 3   33264     5     2
## 4   42783     5     1
## 5   44653     5     1
## 6   51209     5     1
## 7   53355     5     1
## 8   58898     5     1
## 9   64275     5     1
```

Furthermore, for the year data, upon closer inspection we see our previous graph showing yearly averages is somewhat misleading.



When we introduce size, we get a nice constellation as well as insight to help us further refine our model.

Introduce lambdas to penalize effects based on little data.

We will use the same variables in our calculation, but, we will do so by using regularization, where we add a penalty term to our equation in order to reduce the effect of estimates that are obtained from relatively small samples. This way we don't blindly assume a movie with an average rating of 5 is accurate or helpful to our model when it has only been reviewed by one person.

By introducing a lambda into the formula, an effect that had a denominator of 1 will be greatly weakened, while an effect that had a large denominator of 100 will not be greatly changed. We'll start with a lambda of 7, for luck.

```
lambda <- 7
movie_avgs_reg <- train_set %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu)/(n()+lambda))

user_avgs_reg <- train_set %>%
     left_join(movie_avgs_reg, by='movieId') %>%
     group_by(userId) %>%
     summarize(b_u = sum(rating - mu - b_m)/(n()+lambda))

genre_avgs_reg <- train_set %>%
    left_join(movie_avgs_reg, by='movieId') %>%
    left_join(user_avgs_reg, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_m - b_u)/(n()+lambda))

year_avgs_reg <- train_set %>%
    mutate(year = as.numeric(str_sub(title,-5,-2))) %>%
    left_join(movie_avgs_reg, by='movieId') %>%
    left_join(user_avgs_reg, by='userId') %>%
    left_join(genre_avgs_reg, by='genres') %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - mu - b_m - b_u - b_g)/(n()+lambda))

predicted_ratings <- test_set %>%
    mutate(year = as.numeric(str_sub(title,-5,-2))) %>%
    left_join(movie_avgs_reg, by='movieId') %>%
    left_join(user_avgs_reg, by='userId') %>%
    left_join(genre_avgs_reg, by='genres') %>%
    left_join(year_avgs_reg, by='year') %>%
    mutate(pred = mu + b_m + b_u + b_g + b_y) %>%
    .$pred

RMSE(test_set$rating, predicted_ratings)
```

```
## [1] 0.8648871
```

RMSE: 0.8648

With our current approach, the only thing left to do is try tuning the lambda parameter to find the value which minimized our errors.

```
lambdas <- c(1:10)

rmses <- sapply(lambdas, function(lambda){
movie_avgs_reg <- train_set %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu)/(n()+lambda))

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

```
genre_avgs_reg <- train_set %>%
    left_join(movie_avgs_reg, by='movieId') %>%
    left_join(user_avgs_reg, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - b_m - b_u)/(n()+lambda))

year_avgs_reg <- train_set %>%
    mutate(year = as.numeric(str_sub(title,-5,-2))) %>%
    left_join(movie_avgs_reg, by='movieId') %>%
    left_join(user_avgs_reg, by='userId') %>%
    left_join(genre_avgs_reg, by='genres') %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - mu - b_m - b_u - b_g)/(n()+lambda))

predicted_ratings <- test_set %>%
    mutate(year = as.numeric(str_sub(title,-5,-2))) %>%
    left_join(movie_avgs_reg, by='movieId') %>%
    left_join(user_avgs_reg, by='userId') %>%
    left_join(genre_avgs_reg, by='genres') %>%
    left_join(year_avgs_reg, by='year') %>%
    mutate(pred = mu + b_m + b_u + b_g + b_y) %>%
    .$pred

RMSE(test_set$rating, predicted_ratings)
})

lambdas[which.min(rmses)]
```

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

```
rmses[which.min(rmses)]
```

```
## [1] 0.8647994
```

Here we see the best lambda to use is 5, not too far off from our initial lambda of 7. With this our errors have reduced slightly to an RMSE of 0.8648. We could possibly get an even better result if we used a more fine grained lambda.

Results

Lets see what happens when we train using the full edx dataset and test against our validation set. We'll just use a couple lambdas to see if there is a trend.

```
mu <- mean(edx$rating)
lambdas <- c(4:6)

rmses <- sapply(lambdas, function(lambda){
movie_avgs_reg <- edx %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu)/(n()+lambda))

user_avgs_reg <- edx %>%
     left_join(movie_avgs_reg, by='movieId') %>%
```

```
     group_by(userId) %>%
     summarize(b_u = sum(rating - mu - b_m)/(n()+lambda))

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

year_avgs_reg <- edx %>%
    mutate(year = as.numeric(str_sub(title,-5,-2))) %>%
    left_join(movie_avgs_reg, by='movieId') %>%
    left_join(user_avgs_reg, by='userId') %>%
    left_join(genre_avgs_reg, by='genres') %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - mu - b_m - b_u - b_g)/(n()+lambda))

predicted_ratings <- validation %>%
    mutate(year = as.numeric(str_sub(title,-5,-2))) %>%
    left_join(movie_avgs_reg, by='movieId') %>%
    left_join(user_avgs_reg, by='userId') %>%
    left_join(genre_avgs_reg, by='genres') %>%
    left_join(year_avgs_reg, by='year') %>%
    mutate(pred = mu + b_m + b_u + b_g + b_y) %>%
    .$pred

RMSE(validation$rating, predicted_ratings)
})

lambdas[which.min(rmses)]
```

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

```
rmses[which.min(rmses)]
```

```
## [1] 0.8642929
```

We see a lambda of 5 gives us an **RMSE of 0.8642929**

Since we used lambdas of 4, 5 and 6 we know that 4 and 6 but have produced higher RMSEs, so we are at the best spot. Again, this could possibly be further refined by looking at more fine grained lambdas.

Conclusion

By leveraging the additional variables in the dataset we were able to construct a prediction system that allows us to guess with decent accuracy what a movie will be rated given knowledge of ratings of individual movies, users, years and genre combinations. From this we can find movies that a particular user will likely enjoy and give a high rating, or conversely filter out movies that we think they would not enjoy.

Further Research

Our approach closely matched that of a linear regession formula where we calculate $y = b0 + b1 + b2 + e$, at least in principal. It is likely that more advanced techniques such as Linear Discriminant Analysis (LDA)

could be employed to help us discover more nuanced trends across users and movies. I worked within the constraints of building formulas using more basic R functionality in order to accomodate hardware & software memory limitations, so there is still frontier to be explored for the intrepid data scientist.