Course 9 - Capstone

Robert Clark

5/1/2019

MovieLens Project

Overview

The goal of this project is to build a movie recommendation system that is capable of fairly accurately predicting how a user will rate a movie. To do this, I will use the MovieLens dataset that is publicly available. The dataset consists of ratings given to movies by users. The userId, movieId, rating, timestamp, movie title and the movie's genres are included.

To be able to identify how accurate our predictions are, I will be using the same loss function that the Netflix challenge winners used, which is the residual mean squared error, or RMSE. A RMSE of 0 would mean that every one of our predictions is perfect, so our goal will be to minimize the RMSE. If we define $y_{u,i}$ as the rating of movie i by user u, and $\hat{y}_{i,u}$ as our prediction, then the RMSE can be calculated as:

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

In this equation, N is the number of movie combinations and the sum is occuring over all these combinations. If the RMSE returned from the model is a lot larger than 1, then our predicted rating is usually missing the true rating by one or more stars.

The first step of our project is to load the datasets. The full dataset consists of ~10,000,000 rows. This data will get split into the training set, which we will use to build the model, and the testing set, which will be used for testing the model's accuracy through the RMSE. The training set, called "edx", will hold 90% of all the data, and the testing set, called "validation", will hold the other 10%.

While trying to load and create the edx and validation datasets with the provided code from the class, I ran into technical issues. For the purpose of this project, I have simply downloaded and loaded an already prepared edx and validation datasets.

Create Test and Validation Sets

The first step will be to load the datasets, the libraries, as well as create the RMSE function to be used later.

```
# load libraries
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────── tidyverse 1.2.1 ──
## ✔ ggplot2 2.2.1     ✔ purrr   0.2.5
## ✔ tibble  1.4.2     ✔ dplyr   0.7.5
## ✔ tidyr   0.8.1     ✔ stringr 1.4.0
## ✔ readr   1.1.1     ✔ forcats 0.3.0
```

```
## ── Conflicts ──────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Warning in as.POSIXlt.POSIXct(Sys.time()): unknown timezone 'zone/tz/2019a.
## 1.0/zoneinfo/America/Denver'
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
# load edx and validation sets
edx <- readRDS("/Users/robertwilliamclark/Downloads/edx.rds")
validation <- readRDS("/Users/robertwilliamclark/Downloads/validation.rds")


# create RMSE function
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Investigate Dataset

The next step will be to investigate the data. Proposing some questions and looking for the answers will help me understand the data better. It may even help me brainstorm some relationships between the data that I could try to use during the model building step.

```
# examine the data
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                 Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7        Children|Comedy|Fantasy
```

```
# how many rows and columns are there?
nrow(edx)
```

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

```
ncol(edx)
```

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

```
# how many 0's were given as a rating?
zeros <- edx$rating == 0
sum(zeros)
```

```
## [1] 0
```

```
# how many 3's were given as a rating?
threes <- edx$rating == 3
sum(threes)
```

```
## [1] 2121240
```

```
# What is the distribution of the ratings given?
edx %>% group_by(rating) %>% summarize(count = n(), percent = n() / 9000055)
```

```
## # A tibble: 10 x 3
##    rating   count percent
##     <dbl>   <int>   <dbl>
##  1    0.5   85374 0.00949
##  2    1    345679 0.0384
##  3    1.5  106426 0.0118
##  4    2    711422 0.0790
##  5    2.5  333010 0.0370
##  6    3   2121240 0.236
##  7    3.5  791624 0.0880
##  8    4   2588430 0.288
##  9    4.5  526736 0.0585
## 10    5   1390114 0.154
```

```
# most of the ratings are whole numbers. It is more common for a user to give a whole nu
mber review, instead of, say, a 3.5 or 2.5.

# Sum up the percentages of whole number ratings
edx %>% group_by(rating) %>% summarize(count = n(), percent = n() / 9000055) %>% filter(
rating %in% c(1,2,3,4,5)) %>% summarize(sum = sum(percent))
```

```
## # A tibble: 1 x 1
##     sum
##   <dbl>
## 1 0.795
```

```
# almost 80% of the ratings in the train set are a whole number review. Would a model wo
rk better if the predicted rating was rounded up or down to a whole number?



# how many different movies are there?
edx %>% summarize(n_movies = n_distinct(movieId)) %>% .$n_movies
```

```
## [1] 10677
```

```
# how many different users are there?
edx %>% summarize(n_users = n_distinct(userId)) %>% .$n_users
```

```
## [1] 69878
```

```
# how many ratings per genre?
genres <- c("Comedy", "Romance", "Action", "Crime", "Thriller", "Sci-Fi", "Thriller", "A
dventure", "Children", "Fantasy", "War", "Musical", "Western", "Mystery", "Film-Noir",
"Horror", "Documentary")
smallgenres <- c("Comedy", "Romance", "Action")

genrecount <- sapply(genres, function(x){
  edx %>% filter(str_detect(genres,x)) %>% nrow()
})
genrecount
```

```
##      Comedy     Romance      Action       Crime    Thriller      Sci-Fi
##     3540930     1712100     2560545     1327715     2325899     1341183
##    Thriller   Adventure    Children     Fantasy         War     Musical
##     2325899     1908892      737994      925637      511147      433080
##     Western     Mystery   Film-Noir      Horror Documentary
##      189394      568332      118541      691485       93066
```

```
# which movie has the greatest number of ratings?
maxratings <- edx %>% group_by(title) %>% summarize(count = length(title))
index <- which.max(maxratings$count)
maxratings$title[index]
```

```
## [1] "Pulp Fiction (1994)"
```

```
# find the number of ratings for a specific movie
movie <- "Jurassic Park"
ratings <- edx %>% filter(title == movie) %>% summarize(count = n())
paste("# of ratings for", movie, ":", ratings)
```

```
## [1] "# of ratings for Jurassic Park : 0"
```

```
# what are the five most given ratings in order from most to least
edx %>% group_by(rating) %>% summarize(count = n()) %>% top_n(5) %>% arrange(desc(count)
)
```

```
## Selecting by count
```

```
## # A tibble: 5 x 2
##   rating   count
##    <dbl>   <int>
## 1    4   2588430
## 2    3   2121240
## 3    5   1390114
## 4    3.5  791624
## 5    2    711422
```

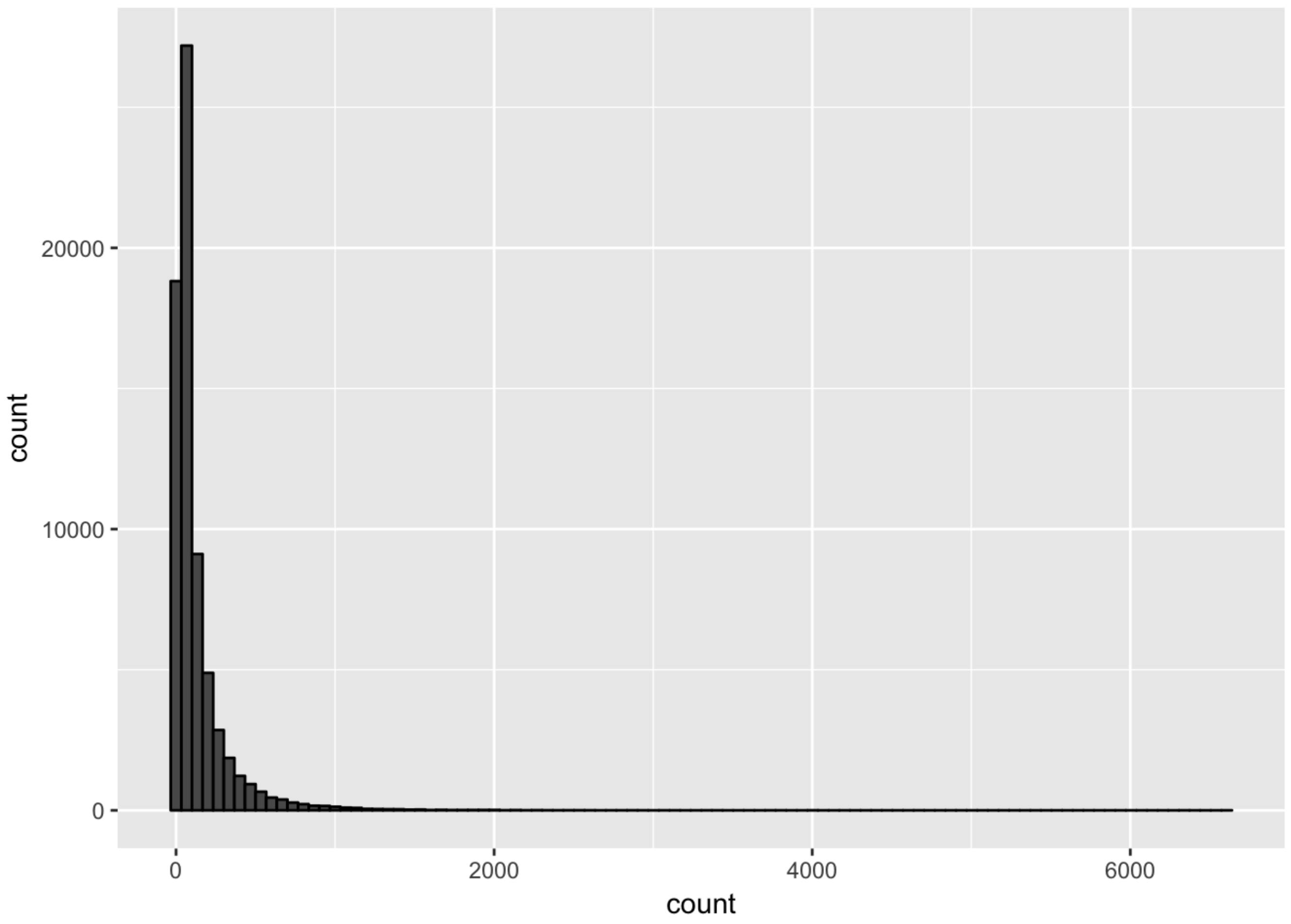
```
# how many users rated 1000 movies or more?
edx %>% group_by(userId) %>% summarize(count = n()) %>% filter(count >= 1000) %>% nrow()
```

```
## [1] 611
```

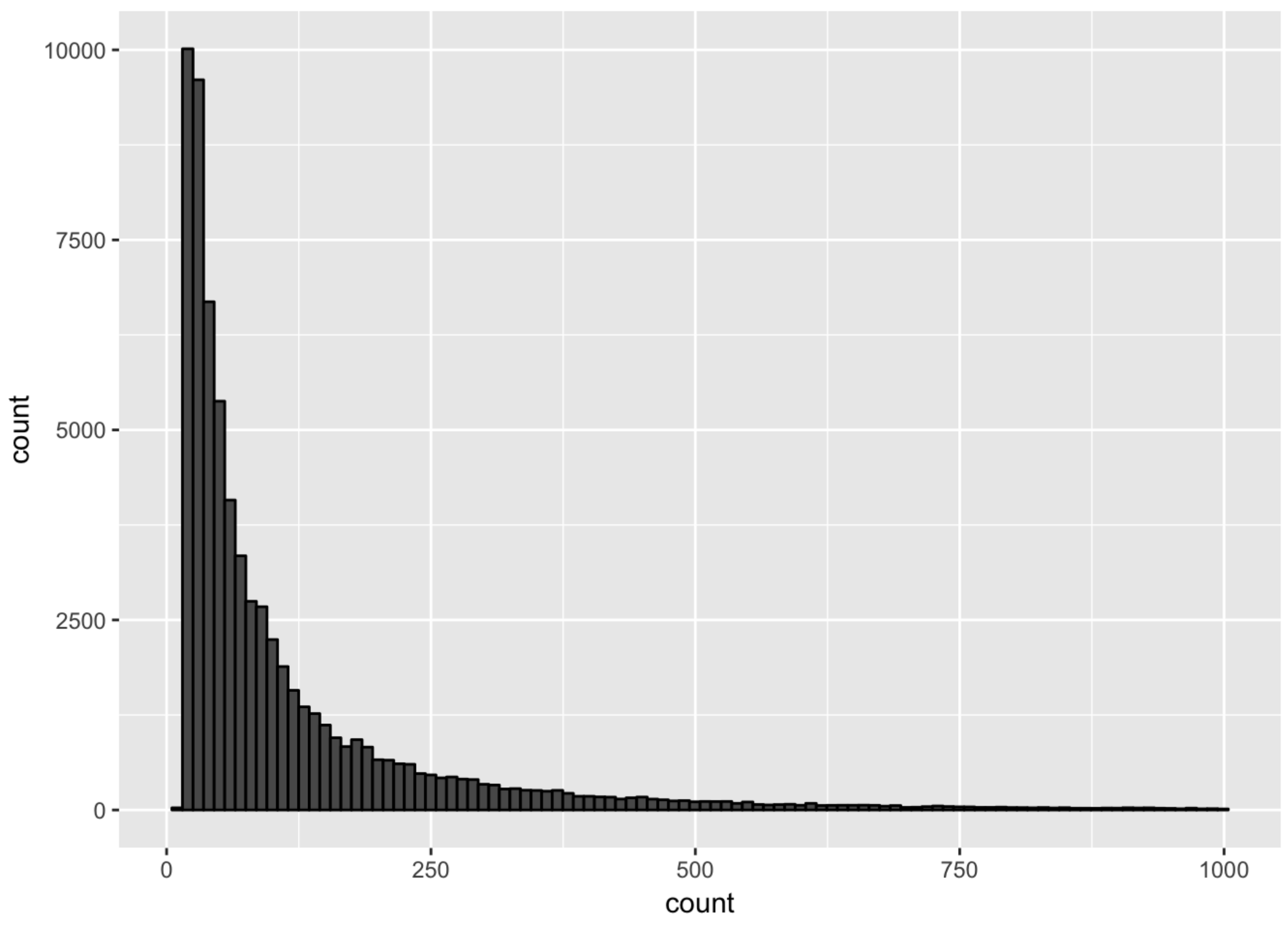
```
# 611, less than 1% of the user base, rated over 1000 movies each.

# What is the total number of ratings of this small group of users?
top_users <- edx %>% group_by(userId) %>% summarize(count = n()) %>% filter(count >= 100
0) %>% mutate(top_user = "yes")
edx %>% left_join(top_users, by='userId') %>% filter(top_user == "yes") %>% nrow()
```

```
## [1] 867198
```

```
# these 611 users together submitted 867198 ratings. That is almost 10% of all ratings!


# What is the distribution of number of movies rated by each user?
edx %>% group_by(userId) %>% summarize(count = n()) %>% qplot(count, geom = "histogram",
bins = 100, data = ., color = I("black"))
```



```
# the top users are making the rest of the chart hard to see. What happens if we remove
 the ratings from these top users?
edx %>% left_join(top_users, by='userId') %>% filter(is.na(top_user)) %>% group_by(userI
d) %>% summarize(count = n()) %>% qplot(count, geom = "histogram", bins = 100, data = .,
color = I("black"))
```



```
# what was the average number of ratings over all users?
num_user_ratings <- edx %>% group_by(userId) %>% summarize(count = n())
mean(num_user_ratings$count)
```

```
## [1] 128.7967
```

Prepare EDX and Validation for Modeling

The next step is to make some final preparations and cleaning on the edx and validation data sets to prepare for modeling. Some of the models I will try later in the project will require data that has not been calculated yet, such as the number of days after a movie's first review the user rated the same movie, referred to as the "review_day". In order to make the computations and the code simpler during the model building stage, I will make these calculations once and add it to the datasets permanently. I will first calculate the number of days after a movie's first review the user put their review in. I will then separate out the year from the movie title, if there is one, and then calculate the decade of that year. This data will need to be added to both the edx and validation datasets.

```
# calculate minimum movie timestamp for edx
min_edx_timestamp <- edx %>%
  group_by(movieId) %>%
  summarize(first_review = min(timestamp))

# calculate the number of days between each review day and the movie's first review day
 and add it to edx
edx <- edx %>% left_join(min_edx_timestamp, by='movieId') %>% mutate(review_day = ceilin
g((timestamp - first_review) / 86400))


# calculate minimum movie timestamp for validation
min_val_timestamp <- validation %>%
  group_by(movieId) %>%
  summarize(first_review = min(timestamp))

# calculate the number of days between each review day and the movie's first review day
 and add it to validation
validation <- validation %>% left_join(min_val_timestamp, by='movieId') %>% mutate(revie
w_day = ceiling((timestamp - first_review) / 86400))



# this code chucnk will split out the movie title and the year, inserting the year into
 a new column, then add the decade to a new column
edx <- edx %>%
  # take off any whitespaces
  mutate(title = str_trim(title)) %>%
  # create title_tmp and year column
  extract(title, c("title_tmp", "year"), regex = "^(.*) \\(([0-9 \\-]*)\\)$", remove = F
) %>%
  # for series take debut date
  mutate(year = if_else(str_length(year) > 4, as.integer(str_split(year, "-", simplify =
T)[1]), as.integer(year))) %>%
  # replace title NA's with original title
  mutate(title = if_else(is.na(title_tmp), title, title_tmp)) %>%
  # drop title_tmp column
  select(-title_tmp)  %>%
  # generic function to turn (no genres listed) to NA
  mutate(genres = if_else(genres == "(no genres listed)", `is.na<-`(genres), genres)) %
>%
  # add decade column
  mutate(decade = year - year %% 10)


# we will need to add the year and decade  to validation as well
validation <- validation %>%
  # take off any whitespaces
  mutate(title = str_trim(title)) %>%
  # create title_tmp and year column
  extract(title, c("title_tmp", "year"), regex = "^(.*) \\(([0-9 \\-]*)\\)$", remove = F
) %>%
  # for series take debut date
  mutate(year = if_else(str_length(year) > 4, as.integer(str_split(year, "-", simplify =
T)[1]), as.integer(year))) %>%
  # replace title NA's with original title
  mutate(title = if_else(is.na(title_tmp), title, title_tmp)) %>%
  # drop title_tmp column
  select(-title_tmp)  %>%
  # generic function to turn (no genres listed) to NA
  mutate(genres = if_else(genres == "(no genres listed)", `is.na<-`(genres), genres)) %
>%
  # add decade column
  mutate(decade = year - year %% 10 )
```

Begin Recommendation Modeling

Now that we have separated the year from the movie title, calculated the decade, and we have calculated the review_day, we are ready to move on to the model building. After reading about the Netflix challenge and the models that the winners implemented, I wanted to explore the different biases that might make up a user's rating of a movie.

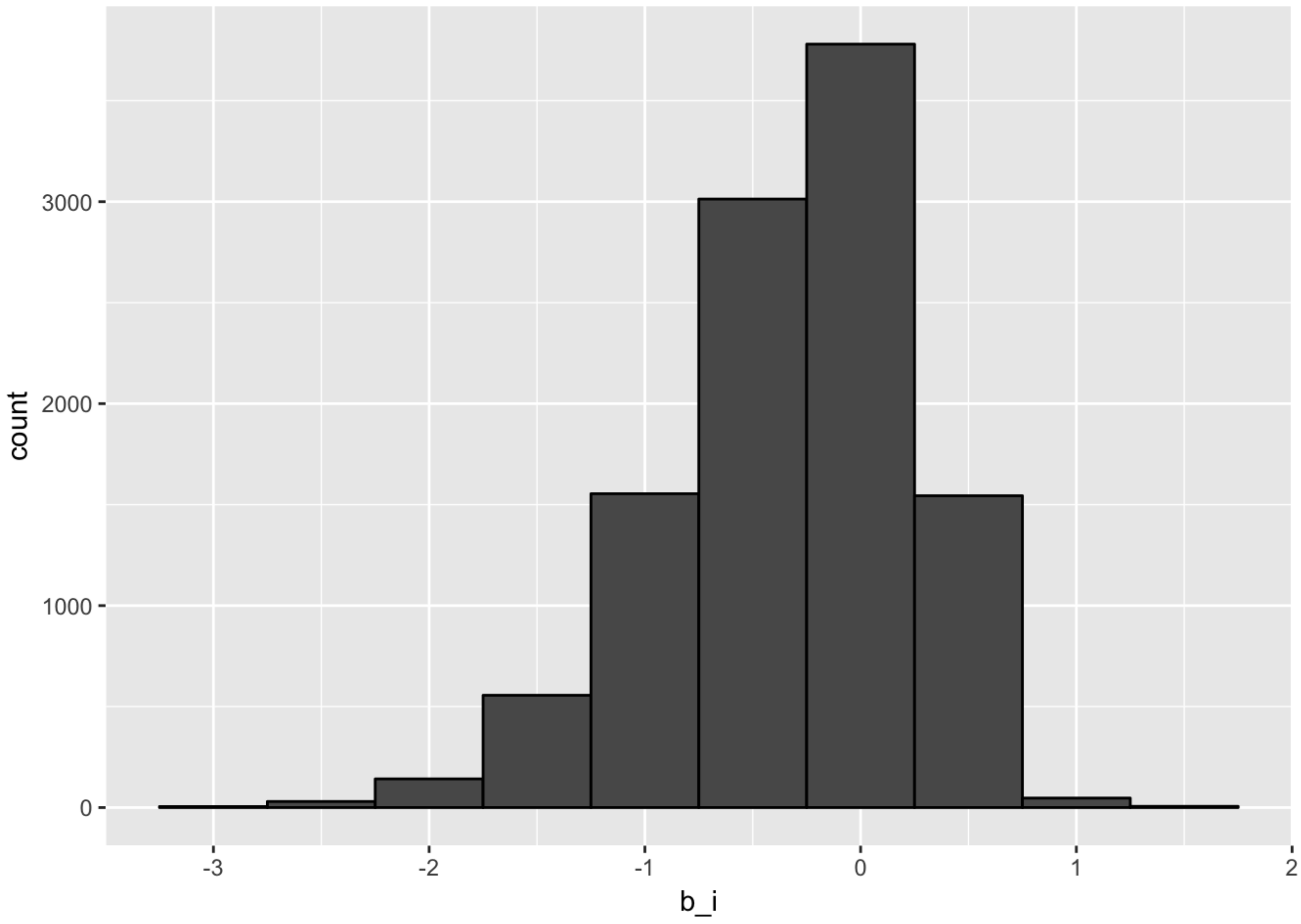
For example, let's say a user rates a movie 2 stars. The starting point for predicting those 2 stars would be the average rating for all movies, which is about 3.5. So what brought the rating down from the average rating to a 2 star rating? Could it be that particular movie usually gets rated lower? Could it be that this particular user tends to give movie ratings a little lower? Could it be that this user doesn't like this genre of movie? It is these biases that I will be testing to see if they can help provide an accurate rating prediction.

The first bias we will add into our predictions is the movie bias, called b_i. This effect is how some movies usually get rated higher or lower than others. When predicting how someone will rate The Shawshank Redemption, it is more accurate to predict a higher rating than predicting how someone will rate Battlefield Earth.

The way we will calculate this is by subtracting the overall rating average, which is μ, from each individual rating, and then finding the average rating for each movie. If, for example, The Shawshank Redemption has an average rating of 4.5 (just a guess), then the b_i for this movie would be 1.

Once calculated, we will join in these terms into our validation dataset, create the predictions, and calculate our RMSE.

```
# calculate the average rating for all movies, which will be the baseline of our predict
ion model
mu <- mean(edx$rating)

# calculate the b_i's, the least squared estimate, aka, the distance away the rating is
 from the mean (3.5 ish)
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# plot the b_i's
movie_avgs %>% qplot(b_i, geom = "histogram", bins = 10, data = ., color = I("black"))
```



```
# join in the movie_avgs and predict the movie ratings by adding the b_i with the overal
l average
predicted_ratings <- validation %>% left_join(movie_avgs, by='movieId') %>% mutate(pred
= mu + b_i) %>% .$pred

# calculate RMSE from these predicted ratings
model_1_rmse <- RMSE(predicted_ratings, validation$rating)
model_1_rmse
```

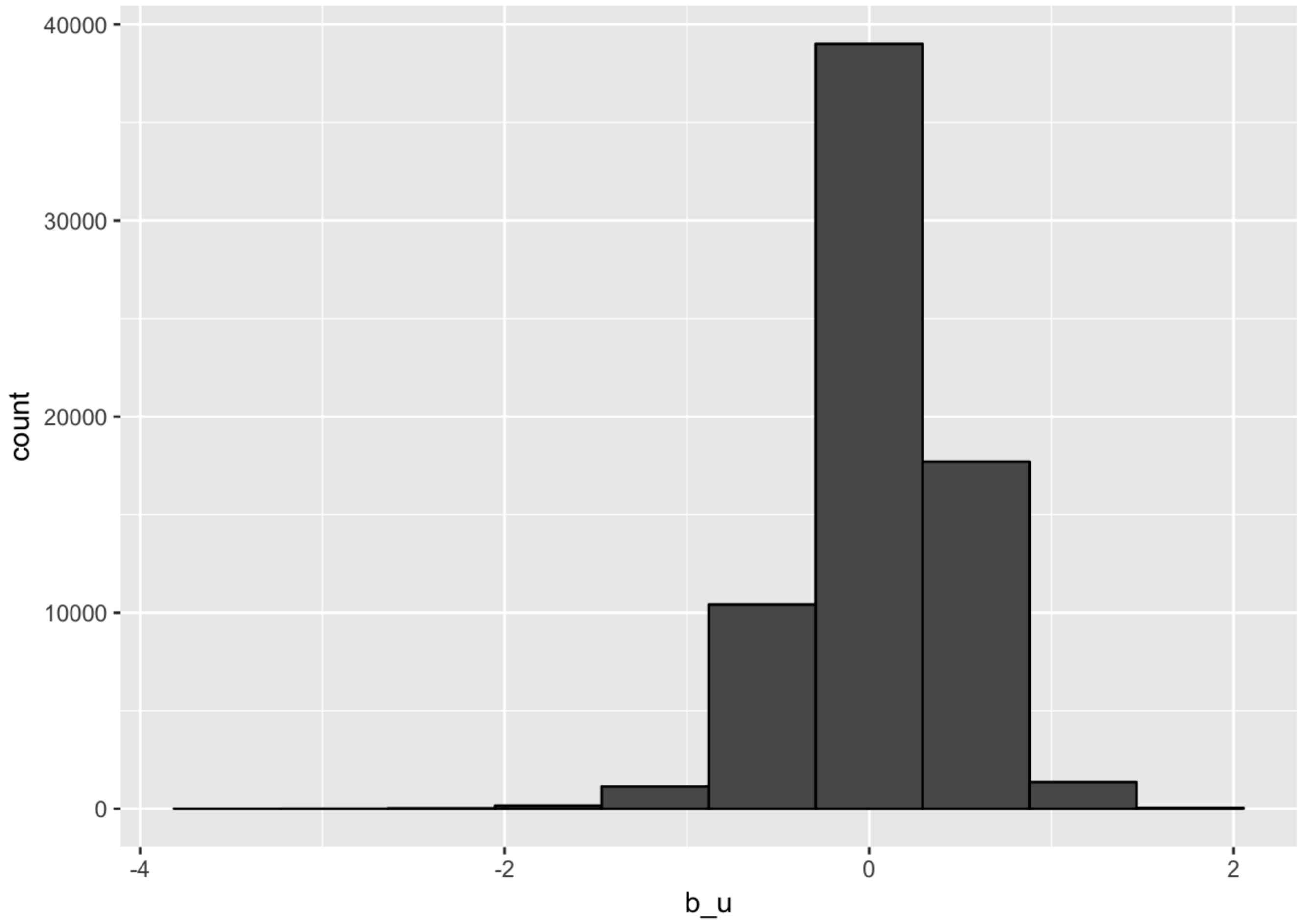
```
## [1] 0.9439087
```

Already we have a RMSE below 1.

The next bias we will add into our predictions is the user bias. There are some users that seem to love everything they see, so they give higher ratings than others. Then there are some users that seem to be hard to please, so they tend to give lower ratings than others.

To calculate these, we will subtract the overall average, mu, and the individual movie average, b_i, from the rating and find the average movie rating for each user. This will be the user bias, denoted as b_u.

```
# calculate the b_u's, the user specific effect.
user_avgs <- edx %>% left_join(movie_avgs, by='movieId') %>% group_by(userId) %>% summar
ize(b_u = mean(rating - mu - b_i))

# plot the b_u's
user_avgs %>% qplot(b_u, geom = "histogram", bins = 10, data = ., color = I("black"))
```



```
# predict the rmse's with the b_i's and b_u's
predicted_ratings <- validation %>% left_join(movie_avgs, by='movieId') %>% left_join(us
er_avgs, by='userId') %>% mutate(pred = mu + b_i + b_u) %>% .$pred

# calculate RMSE from these predicted ratings
model_2_rmse <- RMSE(predicted_ratings, validation$rating)
model_2_rmse
```

```
## [1] 0.8653488
```

That gave the model a good improvement. We are already below an RMSE of .87. I want to add in the other biases to see if we can improve the model even more.

The next bias we will investigate is if there is a bias that comes from how early you review a movie. It is possible that if you review a movie very soon after the movie comes out that it means that you were excited to see the movie. If you were really excited to see the movie, it is possible that you would give a slightly higher rating than you normally would. So if we find a user that rated a movie very soon after it came out, should we predict they will rate it slightly higher than normal?

This is where the review_day will be used, as mentioned above. We will subtract out the overall average, μ, as well as the other biases, to calculate this bias, denoted as b_r.

```
# join in the movie_avgs and user_avgs data sets so that we can find the average rating
 for each review day without the influence of the movie average and the user average.
reviewday_avgs <- edx %>% left_join(movie_avgs, by='movieId') %>% left_join(user_avgs, b
y='userId') %>% group_by(review_day) %>% summarize(b_r = mean(rating - mu - b_i - b_u))

# predict the rmse's with the b_i's, b_u's and b_r's
predicted_ratings <- validation %>% left_join(movie_avgs, by='movieId') %>% left_join(us
er_avgs, by='userId') %>% left_join(reviewday_avgs, by='review_day') %>% mutate(pred = m
u + b_i + b_u + b_r) %>% .$pred

# calculate RMSE from these predicted ratings
model_3_rmse <- RMSE(predicted_ratings, validation$rating)
model_3_rmse
```

```
## [1] 0.8651374
```

The RMSE for this new model is an improvment, but only very slightly so. Since it is still at least an improvement, we'll go ahead and keep this bias in the model.

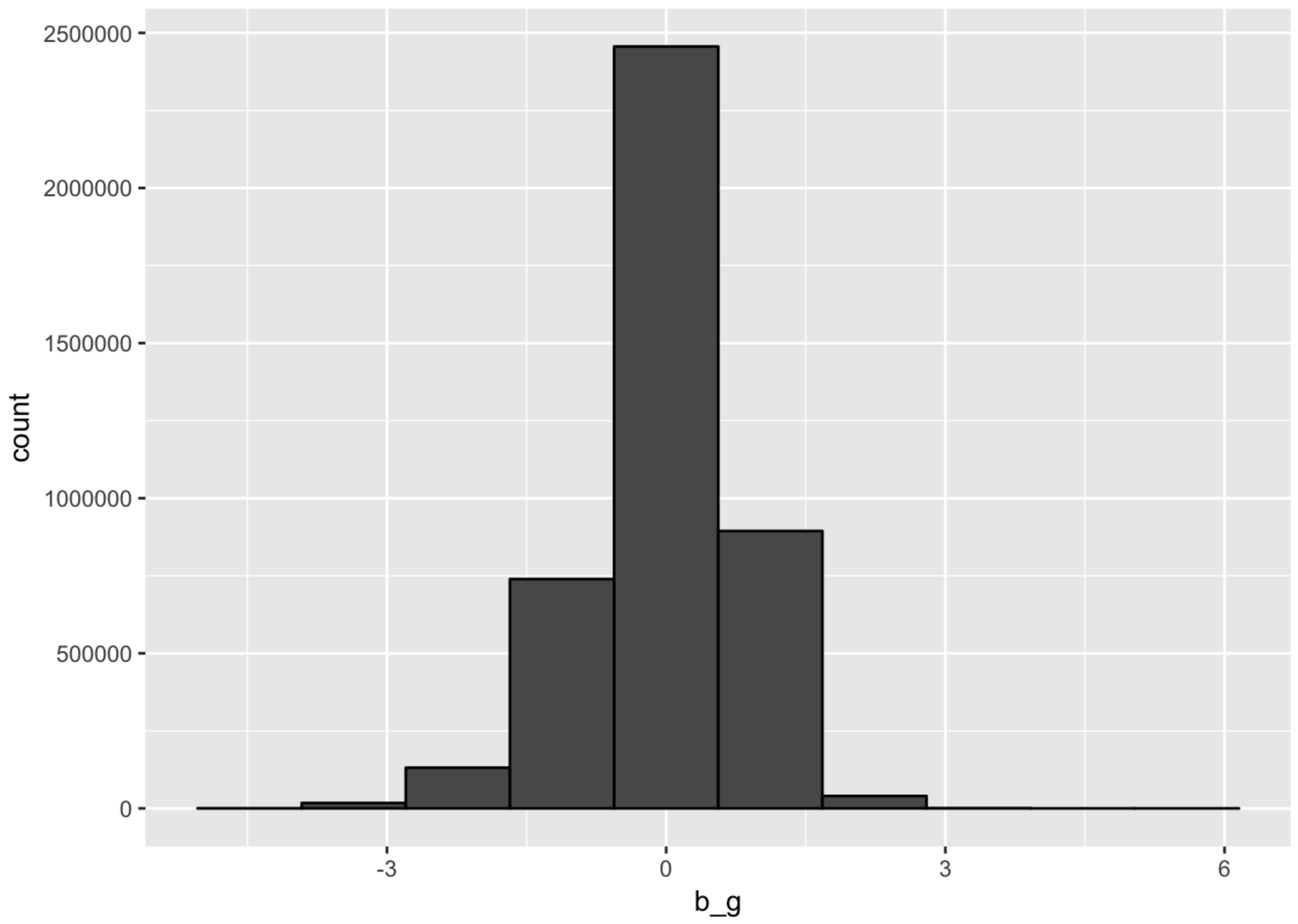
I next want to try and include the genre into our model somehow. I believe that some users tend to enjoy certain genres more than others. So I will try to find the bias for each user and the genres that they have rated. I will be keeping the genres grouped together as they are now, because an Action/Comedy movie is a much different genre than Action/Horror.

```
# find the average that a user rates a genre, without the bias from the movie effect, us
er effect, and review_day effect
user_genre_avgs <- edx %>% left_join(movie_avgs, by='movieId') %>% left_join(user_avgs,
by='userId') %>% left_join(reviewday_avgs, by='review_day') %>% group_by(userId, genres)
%>% summarize(b_g = mean(rating - mu - b_i - b_u - b_r))

# predict the rmse's with the b_i's, b_u's, b_r's and b_g's, putting in a 0 when there i
s no b_g
predicted_ratings <- validation %>% left_join(movie_avgs, by='movieId') %>% left_join(us
er_avgs, by='userId') %>% left_join(reviewday_avgs, by='review_day')  %>% left_join(user
_genre_avgs, by = c("userId", "genres")) %>% mutate(pred = mu + b_i + b_u + b_r + ifelse
(is.na(b_g), 0, b_g)) %>% .$pred

# calculate RMSE from these predicted ratings
model_4_rmse <- RMSE(predicted_ratings, validation$rating)
model_4_rmse
```

```
## [1] 0.9179427
```

This model actually got quite worse. It is probably because in the state the genres are in currently, with so many different genre combinations, it is very likely that a user will have rated a movie from that genre category only once. This would give a lot of weight on those ratings, thus skewing the predictions more than it should. Let's investigate this.

```
# plot the b_g's
user_genre_avgs %>% qplot(b_g, geom = "histogram", bins = 10, data = ., color = I("blac
k"))
```



```
# let's look at our largest residuals and see the breakdown of our model biases
validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(reviewday_avgs, by='review_day')  %>%
  left_join(user_genre_avgs) %>%
  mutate(residual = rating - (mu + b_i + b_u + b_r + ifelse(is.na(b_g), 0, b_g)), pred =
(mu + b_i + b_u + b_r + ifelse(is.na(b_g), 0, b_g)), mu = mu, b_i = b_i, b_u = b_u, b_r
= b_r, b_g = b_g) %>%
  arrange(desc(abs(residual))) %>%
  select(title, rating, pred, residual, mu, b_i, b_u, b_r, b_g) %>% slice(1:10)
```

```
## Joining, by = c("userId", "genres")
##                                title rating        pred  residual       mu
## 1                    Real Cancun, The    4.5 -1.66515154  6.165152 3.512465
## 2  Expelled: No Intelligence Allowed    4.5 -0.92317277  5.423173 3.512465
## 3     Amores Perros (Love's a Bitch)    0.5  5.91613532 -5.416135 3.512465
## 4        12 Monkeys (Twelve Monkeys)    1.0  6.41426598 -5.414266 3.512465
## 5         Terminator 2: Judgment Day    1.0  6.37803526 -5.378035 3.512465
## 6                         Annie Hall    0.5  5.82397462 -5.323975 3.512465
## 7                       Citizen Kane    0.5  5.77881435 -5.278814 3.512465
## 8              Benji: Off the Leash!    4.5 -0.70198026  5.201980 3.512465
## 9                          Gladiator    0.5  5.58346129 -5.083461 3.512465
## 10           History of Violence, A    5.0 -0.06263898  5.062639 3.512465
##            b_i         b_u          b_r        b_g
## 1  -1.68601892 -1.09115265  0.075636916 -2.4760821
## 2  -1.69579853 -0.25293361  0.097245327 -2.5841512
## 3   0.51489579  0.70742479 -0.012321227  1.1936708
## 4   0.36227784 -0.01950131 -0.001665866  2.5606901
## 5   0.41539425 -0.46756277  0.045424781  2.8723138
## 6   0.59007862 -0.40233784  0.006371878  2.1173968
## 7   0.67480181  0.23844484 -0.022537605  1.3756401
## 8  -1.32496520 -0.33840653  0.065556075 -2.6166298
## 9   0.42450989  0.73648573 -0.028418637  0.9384191
## 10  0.09982875  0.04051662 -0.015125984 -3.7003236
```

The user-genre effect is definitely way too large. This is causing our ratings to go under 0 and over 5, giving us really big residuals. We need to regularize the b_g's if we are going to be able to use those in our model.

```
# our first step should be to use cross-fertilization to choose the lamba that minimizes the rmse
user_genre_sum <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(reviewday_avgs, by='review_day') %>%
  group_by(userId, genres) %>%
  summarize(s = sum(rating - mu - b_i - b_u - b_r), n_i = n())
head(user_genre_sum)
```

```
## # A tibble: 6 x 4
## # Groups:   userId [1]
##   userId genres                              s   n_i
##    <int> <chr>                           <dbl> <int>
## 1      1 Action|Adventure|Drama|Sci-Fi -0.0407     1
## 2      1 Action|Adventure|Sci-Fi       -0.0529     1
## 3      1 Action|Comedy                  0.340      1
## 4      1 Action|Comedy|Crime|Thriller   0.549      1
## 5      1 Action|Comedy|War              0.395      1
## 6      1 Action|Crime|Thriller          0.161      1
```

```
# I tried to use the sapply function to try multiple lambda values to see which one mini
mizes the RMSE, but for some reason it just wouldn't work for me. It kept giving me this
error: "Error in tbl_vars(y) : argument "y" is missing, with no default". So instead, I
 just manually tried multiple lambda values individually. It looked like really high val
ues were giving me the lowest RMSE, so I will go with a lambda value of 10.

# use the regularized b_g's in our prediction model
predicted_ratings <- validation %>%
    left_join(user_genre_sum, by = c("userId", "genres")) %>%
    left_join(movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    left_join(reviewday_avgs, by='review_day') %>%
    mutate(b_g = ifelse(is.na(s), 0, s)/(ifelse(is.na(n_i), 0, n_i)+10)) %>%
    mutate(pred = mu + b_i + b_u + b_r + b_g) %>%
    pull(pred)
model_5_rmse <- RMSE(predicted_ratings, validation$rating)
model_5_rmse
```

```
## [1] 0.8559072
```

The regularization worked out great. This reduced the large biases introduced from users that had only seen 1 movie from a specific genre. Not only did we get back to where our RMSE had been, but it also lowered the RMSE further.

The only bias left that I want to try is to see if there is a bias against older movies. Some people might rate movies from the 50’s lower than movies from the last decade.

```
# is there a bias towards older movies? Let's do a movie decade bias
decade_avgs <- edx %>% left_join(movie_avgs, by='movieId') %>% left_join(user_avgs, by=
'userId') %>% left_join(reviewday_avgs, by='review_day') %>% left_join(user_genre_sum, b
y = c("userId", "genres")) %>% mutate(b_g = ifelse(is.na(s), 0, s)/(ifelse(is.na(n_i), 0
, n_i)+10)) %>% group_by(decade) %>% summarize(b_d = mean(rating - mu - b_i - b_u - b_r
- b_g))

# use decade_avgs to build model 6
predicted_ratings <- validation %>% left_join(movie_avgs, by='movieId') %>% left_join(us
er_avgs, by='userId') %>% left_join(reviewday_avgs, by='review_day')  %>% left_join(user
_genre_sum, by = c("userId", "genres")) %>% left_join(decade_avgs, by='decade') %>% muta
te(b_g = ifelse(is.na(s), 0, s)/(ifelse(is.na(n_i), 0, n_i)+10)) %>% mutate(pred = round
(mu + b_i + b_u + b_r + b_g + ifelse(is.na(b_d), 0, b_d))) %>% .$pred

# calculate RMSE from these predicted ratings
model_6_rmse <- RMSE(predicted_ratings, validation$rating)
model_6_rmse
```

```
## [1] 0.9024327
```

This model got worse again. We will just leave this bias out of the model.

Final Results

So our final model includes these elements forming up the prediction:

Bias	Description
mu	Average rating for all movies
b_i	Movie bias, some movies get rated higher or lower than others
b_u	User bias, some users rate movies higher or lowers than others
b_r	Review day bias, where reviews going up sooner rather than later might get rated differently
b_g	User genre bias, where some users like certain genres more than others. This bias was regularized.

Our predictions can be written as the equation below:

$$\hat{Y}_{u,i} = \mu + b_i + b_u + b_r + b_g$$

Using this equation, and using the RMSE function listed at the beginning of this project, I was able to achieve an RMSE of 0.8559072.

Conclusions

In this project, I have developed a recommendation system taking into account a baseline rating with multiple effect predictors (biases) added in. There was only one bias that I tried, the decade_bias, that did not improve the model. It is possible that if I tried implementing regularization on the decade_bias that it would have yielded better results, but I decided to not include it. The ones that were most influential on the RMSE were the movie bias, the user bias, and the genre bias.

```
## Joining, by = c("userId", "genres")
```

Here is an example of how our model predicts ratings. In the validation set, user 259 reviewed Jurassic Park and gave it a 4 star rating. Using our model to predict what he would have given Jurassic Park had we not known, the elements of our model provided these numbers:

μ: 3.5124652

b_i: 0.1510566 This movie tends to get rated higher than the movie average

b_u: -0.0149962 This user tends to rate movies slightly lower than other users

b_r: -0.0314906 This rating came a little over 8 years after the movie came out, lowering the prediction slightly

b_g: 0.0961473 This user tends to rate movies of this genre higher than others, suggesting this might be a favorite genre of this user

So we have 3.5124652 + 0.1510566 - 0.0149962 - 0.0314906 + 0.0961473, which gives us 3.7131823. Our prediction is farily close, only 0.2868177 off, which is our residual.

Obviously this example had a closer prediction than some of our others. This is just an explanation of how our different biases work together to create the rating predictions. If we looked at some of our higher residuals, we would see that we were still fairly far off on some ratings. This means there might still be some bias that we are not accounting for. The equation for the actual movie ratign could be written like this:

$$Y_{u,i} = \mu + b_i + b_u + b_r + b_g + \epsilon_{u,i}$$

In this equation, epsilon represents independent errors. This error term could hide some bias that we have not yet accounted for. If we were to explore other machine learning models such as matrix factorization or collaborative neighbors, we could find some other interactions as well.

For now, we have achieved our goal of building a movie recommendation system that has produced a relatively low RMSE of 0.8559072.