Movie Recommender

Thomas Linnell

7/25/2019

Introduction

Overview

The goal of this project is to create a recommendation system that will predict with high accuracy the rating that a customer would give to any particular movie that they would watch. This would be useful to organizations that provide movies as a service to help guide customers to select movies that they would like to see, and to avoid selecting movies that they would not.

The information that we have available to us is a large database of movie ratings. However, this database is sparse in that we don't have ratings for all movies by all viewers. The challenge is to be able to use the information available to design a system that would be able to provide a good estimate of a particular users rating of a movie that they haven't rated yet.

It turns out that this is a fairly complicated problem to solve, and involves much more that just providing the average review rating from all previous viewers of the movie. There are many factors to consider - some people like different types of movies more than others, some movies are more liked than others by the general population, the user may be swayed by the actors featured in a movie or the fact that all their friends are talking about a movie, and so on.

Because of all these factors, we have chosen to use a machine learning algorithm approach to solving this problem. This will allow us to rapidy modify our modeling approaches utilizing a wide range of available techniques, and to iteratively add components to our model and to be able to test the effects of these on the overall accuracy.

The goal of this project is to reduce the estimate error to a number below 0.9 RMSE, with increasing project scores as this metric approaches 0.8649.

Project setup and environment

Check for required packages and install when necessary.

```
## Loading required package: tidyverse
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
## -- Attaching packages ----------------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.1.1     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.1
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
## -- Conflicts -------------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

## Loading required package: caret

## Loading required package: lattice

##
## Attaching package: 'caret'

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

## Loading required package: data.table

##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last

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

Data

This project will utilize the dataset *MovieLens 10M*, provided by the GroupLens project, and can be found here

This dataset contains 10 million ratings of approximately 10,000 movies rated by almost 72,000 users.

We will load the data and adjust it into the format that we want to use to analyze the ratings:

```r
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")
```

The structure of this dataset is as follows:

```
glimpse(movielens)
```

```
## Observations: 10,000,054
## Variables: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ movieId   <dbl> 122, 185, 231, 292, 316, 329, 355, 356, 362, 364, 37...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5...
## $ timestamp <int> 838985046, 838983525, 838983392, 838983421, 83898339...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Dumb & Dumbe...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Comedy",...
```

We see that the observations are provided with keys for identifying users and movies, the ratings which are on a scale of 0.5 to 5, titles, classification information(genres) and a timestamp of when the rating was given.

The summary below also provides us with some useful information in that there are no NA's in the data that need to be tidied, and the ratings data do not have any outliers outside of the expected range.

```
summary(movielens)
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18123   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35741   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53608   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title               genres
##  Length:10000054    Length:10000054
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

Test and Validation sets

To begin with, we take the movielens 10M database provided and split this into a test set (90%) and a validation set (10%). Our methodology will be developed on the test set, and measured on the validation set, using the RMSE metric to judge and compare the results as noted above.

```
# Validation set will be 10% of MovieLens data

# for R 3.6 , use this sampling. If using R 3.5 or earlier, use `set.seed(1)` instead
set.seed(1, sample.kind="Rounding")

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)

# Clean up data structures no longer needed

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

Methods and analysis

From the summaries, we see that the data looks like it is in good shape for analysis. We can now proceed to analyzing these data to come up with an accurate prediction methodology. From the reference literature and coursework that we recently completed, we know that our model will require at least a computation of the following:

- A baseline rating (e.g., the mean of all ratings)
- A movie-specific effect (e.g., how this movie is rated vs. all of the other movies)
- A user-specific effect (similar to the above, how this user compares to the others)

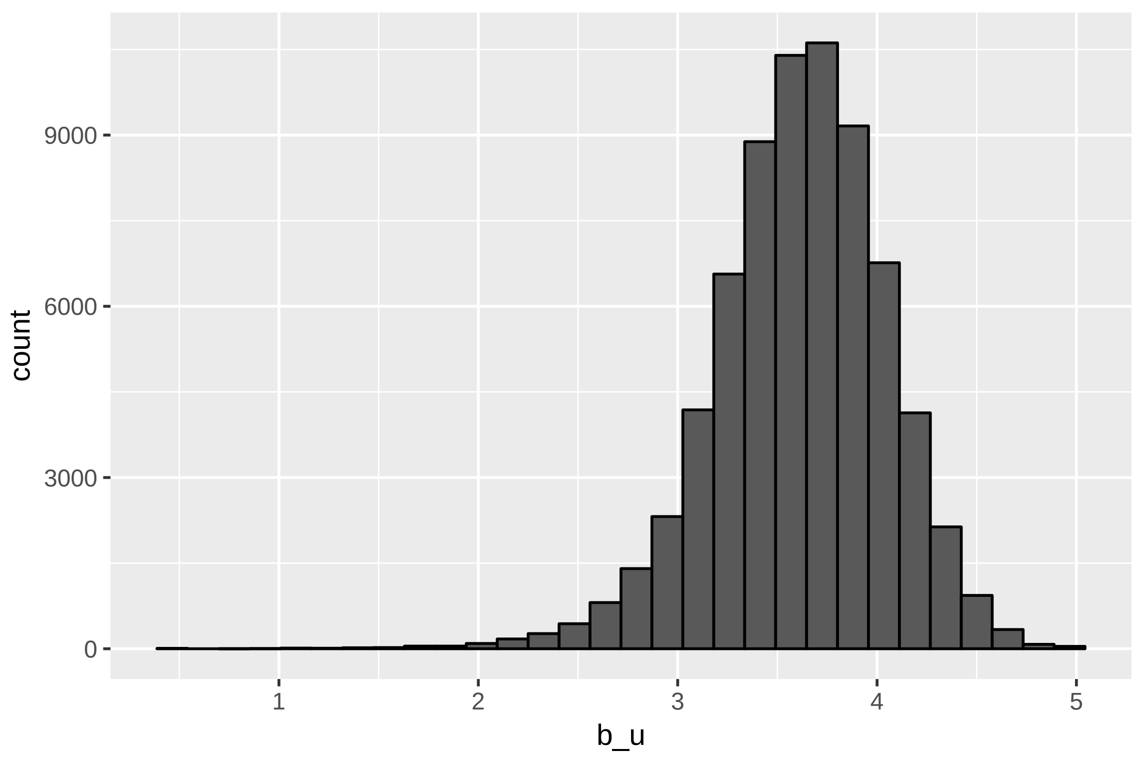
In addition, it may be helpful to regularize the ratings data to reduce the noisiness of movie ratings with low sampling counts. We will use this technique and compare the results to the unmodified dataset.

Baseline Calculation

We start by building a function to compute the RMSE of our model against a test data set, and then use that to compute a baseline RMSE of the mean of all ratings:

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

Now we look at a distribution of the ratings, to get a feel for how the data are distributed:



It appears from this that the data is reasonably normally distributed, with a mean somewhere in the vicinity of 3.5 We next calculate the mean and plug that into our RMSE function:

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

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

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

```
## [1] 1.061202
```

This mean agrees with the calculation that we provided in the summary, above, as expected. Even though this is accurate, we see that the baseline RMSE, being over 1, tells us that a system built with just this calculation will not be very accurate, and will miss the true rating by about one entire star on average. We would like to see if it is possible to reduce this error to something less than one star.

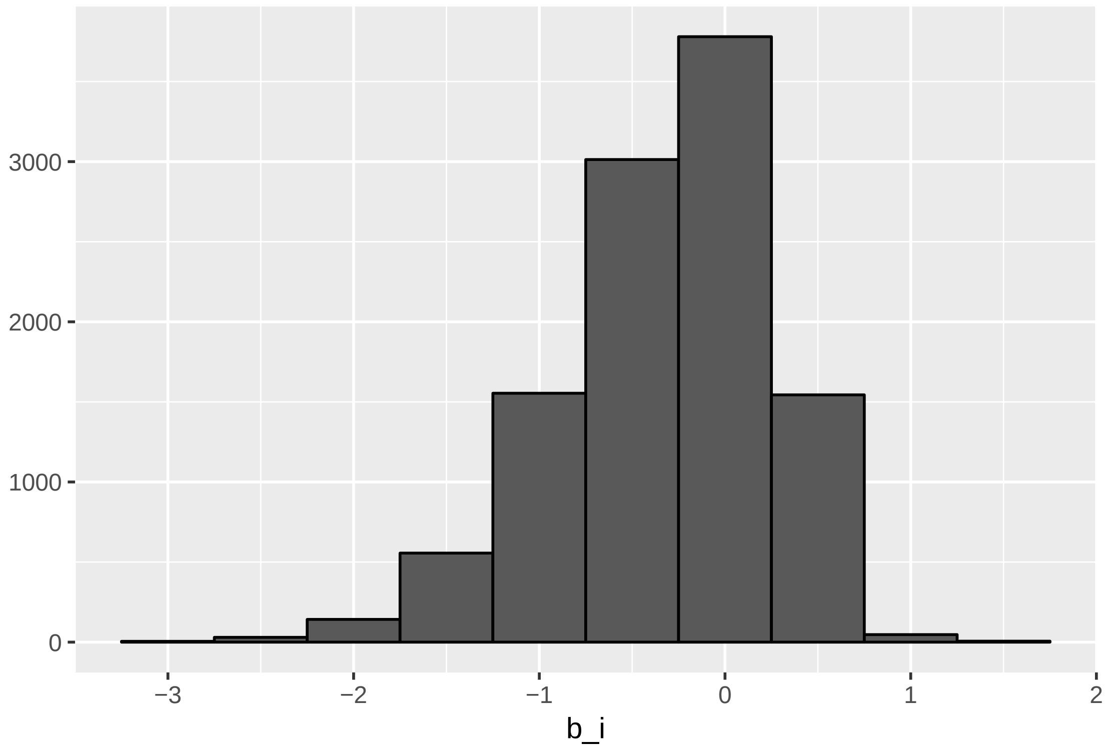
Movie Effects

We now move on to computing a movie-specific effect. In order to compute this efficiently, we use the technique described in the course where the RMSE is calculated over the difference between the rating and our estimate of mu.

First we separate out the value of each rating for each movie less the overall baseline average that we just calculated, and calculate the mean of this value for each movie:

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))
```

We next plot the distribution of the movie effect:



Interestingly, the distribution implies that for most movies the adjustment due to the movie effect will be to lower the rating from the average.

We then use these values to plug in a predicted value for a user rating by applying our baseline mean and our new movie mean calculation:

```
predicted_ratings_1 <- mu_hat + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
```

Then we can compare our predictions to the actual ratings in our validation set for a new calculation of our estimate error:

```
(movie_rmse <- RMSE(predicted_ratings_1, validation$rating))
```

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

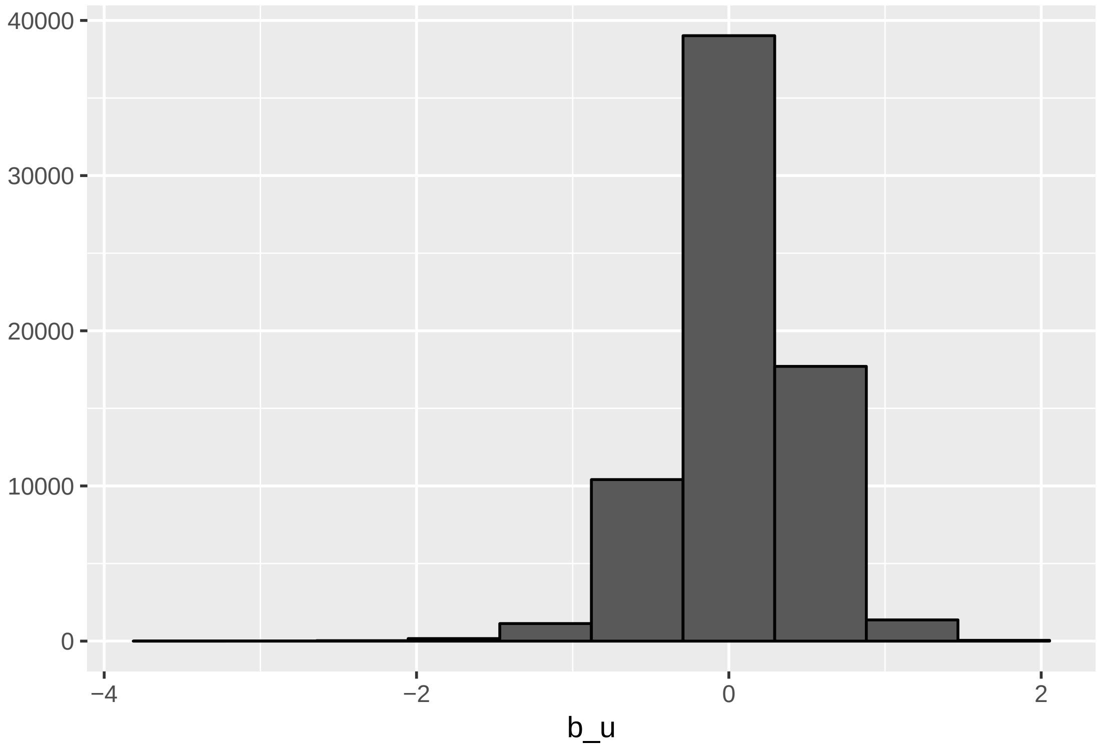
We see that the estimate has improved, so now we will move on to computing the effect due to users, using a similar methodology.

User Effects

Again we are going to use the technique of approximating the estimate of user effect by computing the average of the difference of the rating and our estimate of the mean and estimate of the movie effect that we just computed:

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

If we take a look at the distribution due to the user effect, we see a very different distribution than what we saw for the movie-related effect. Here we see that the effect is mainly slightly positive:



Once we have done that, we can plug this value in as before along with our baseline and movie effect estimate to arrive at a prediction of the rating for each movie:

```
predicted_ratings_2 <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  pull(pred)
```

And we see what effect the user variation has on our prediction:

```
(user_rmse <- RMSE(predicted_ratings_2, validation$rating))
```

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

This clearly has improved our prediction.

Regularization Effect

By combining movie and user effects, we have arrived at a much smaller RMSE. But can we do better? We have not yet explored whether removing the noise from ratings of movies with low sample counts will improve our score, so we will take another look at the data.

If we examine our predictions, taking into account the baseline and movie effects versus the actual user ratings, we see that there are still significant variations, as shown by the calculation below:

```
validation %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(residual = rating - (mu_hat + b_i)) %>%
  arrange(desc(abs(residual))) %>%
  select(title,  residual) %>% distinct() %>% slice(1:15)
```

```
##                                                                   title
## 1                                                 PokÃ©mon Heroes (2003)
## 2                                       Shawshank Redemption, The (1994)
## 3                                                  Godfather, The (1972)
## 4                                             Usual Suspects, The (1995)
## 5                                                Schindler's List (1993)
## 6                   Pokemon 4 Ever (a.k.a. PokÃ©mon 4: The Movie) (2002)
## 7                                                      Casablanca (1942)
## 8                                                     Rear Window (1954)
## 9                                                  Third Man, The (1949)
## 10                             Seven Samurai (Shichinin no samurai) (1954)
## 11                                        Godfather: Part II, The (1974)
## 12                                   Faces of Death: Fact or Fiction? (1999)
## 13                                               Dark Knight, The (2008)
## 14 Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)
## 15                                  One Flew Over the Cuckoo's Nest (1975)
##     residual
## 1   3.970803
## 2  -3.955131
## 3  -3.915366
## 4  -3.865854
## 5  -3.863493
## 6   3.821782
## 7  -3.820424
## 8  -3.818652
## 9  -3.811426
## 10 -3.806744
## 11 -3.801971
## 12  3.801724
## 13 -3.797068
## 14 -3.795333
## 15 -3.793261
```

If we sort the data by best user rating, we can see that they all have low numbers of reviews. To do this, we need to combine our user ratings with the titles of the movies to which they correspond:

```
movie_titles <- movielens %>%
  select(movieId, title) %>%
  distinct()
```

Going back to our test set, we can look at the movies with the best movie effect scores. We can now see that these are all fairly obscure, and have low ratings counts.

```r
edx %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10)
```

```
## Joining, by = "movieId"
```

```
## # A tibble: 10 x 3
##    title                                                           b_i     n
##    <chr>                                                         <dbl> <int>
##  1 Hellhounds on My Trail (1999)                                  1.49     1
##  2 Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)                            1.49     2
##  3 Shadows of Forgotten Ancestors (1964)                          1.49     1
##  4 Fighting Elegy (Kenka erejii) (1966)                           1.49     1
##  5 Sun Alley (Sonnenallee) (1999)                                 1.49     1
##  6 Blue Light, The (Das Blaue Licht) (1932)                       1.49     1
##  7 Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko~   1.24     4
##  8 Human Condition II, The (Ningen no joken II) (1959)            1.24     4
##  9 Human Condition III, The (Ningen no joken III) (1961)          1.24     4
## 10 Constantine's Sword (2007)                                     1.24     2
```

And when we examine the movies with the worst scores, we also see a similar pattern:

```r
edx %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i, n) %>%
  slice(1:10)
```

```
## Joining, by = "movieId"
```

```
## # A tibble: 10 x 3
##    title                                         b_i     n
##    <chr>                                       <dbl> <int>
##  1 Besotted (2001)                             -3.01     2
##  2 Hi-Line, The (1999)                         -3.01     1
##  3 Accused (Anklaget) (2005)                   -3.01     1
##  4 Confessions of a Superhero (2007)           -3.01     1
##  5 War of the Worlds 2: The Next Wave (2008)   -3.01     2
##  6 SuperBabies: Baby Geniuses 2 (2004)         -2.72    56
##  7 Hip Hop Witch, Da (2000)                    -2.69    14
##  8 Disaster Movie (2008)                       -2.65    32
##  9 From Justin to Kelly (2003)                 -2.61   199
## 10 Criminals (1996)                            -2.51     2
```

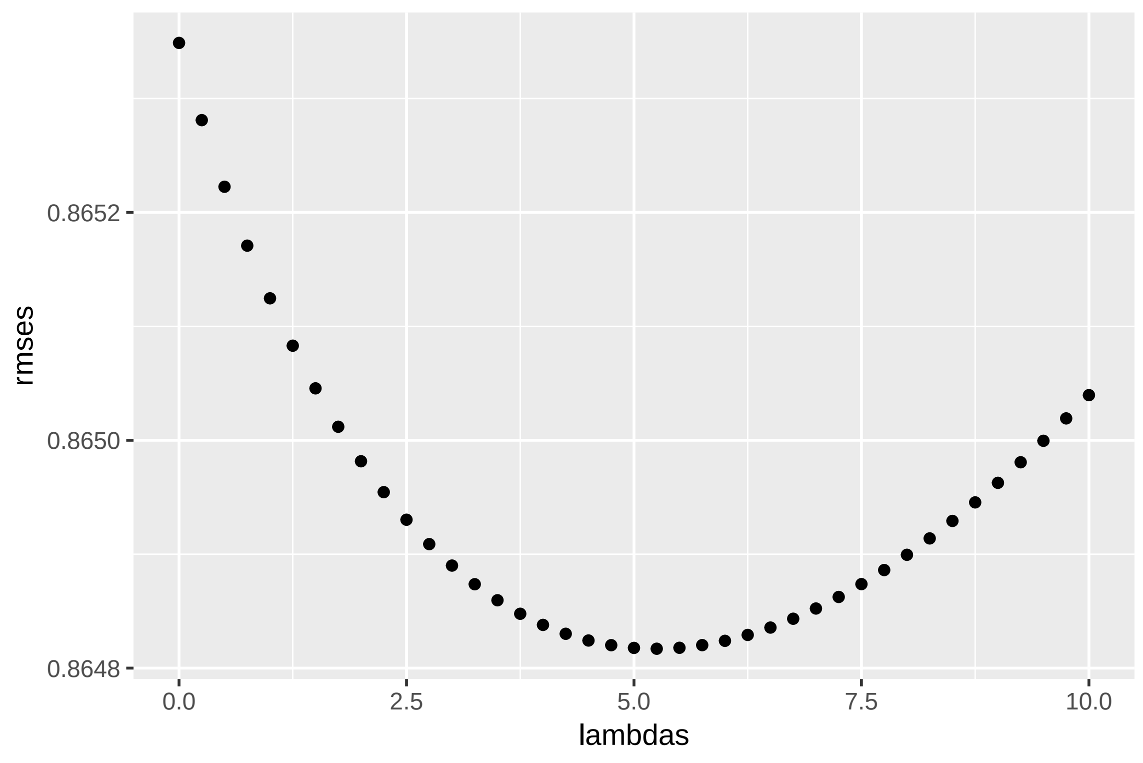
Penalty Calculation

As we see from these lists, most of the movies generating the highest and lowest scores are from a set of movies which have few ratings. Our next step in the process will be to regularize the data to reduce the weighting that is given to these infrequently rated movies. We will use penalized least squares, which adds a penalty term which increases with the value of b_i and also varies inversely with the number of ratings. We will also use cross-validation to test a range of penalty terms to see which value best minimizes our RMSE term:

```r
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){

  mu <- mean(edx$rating)

  b_i_l <- edx %>%
    group_by(movieId) %>%
    summarize(b_i_l = sum(rating - mu)/(n()+l))

  b_u_l <- edx %>%
    left_join(b_i_l, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u_l = sum(rating - b_i_l - mu)/(n()+l))

  predicted_ratings_l <-
    validation %>%
    left_join(b_i_l, by = "movieId") %>%
    left_join(b_u_l, by = "userId") %>%
    mutate(pred = mu + b_i_l + b_u_l) %>%
    pull(pred)

  return(RMSE(predicted_ratings_l, validation$rating))
})
```

When we plot the values of RMSE from our calculations, we see that it does converge to a minima within the range that we selected. We now calculate the best penalty term and save that for the next step.

```r
qplot(lambdas, rmses)
```



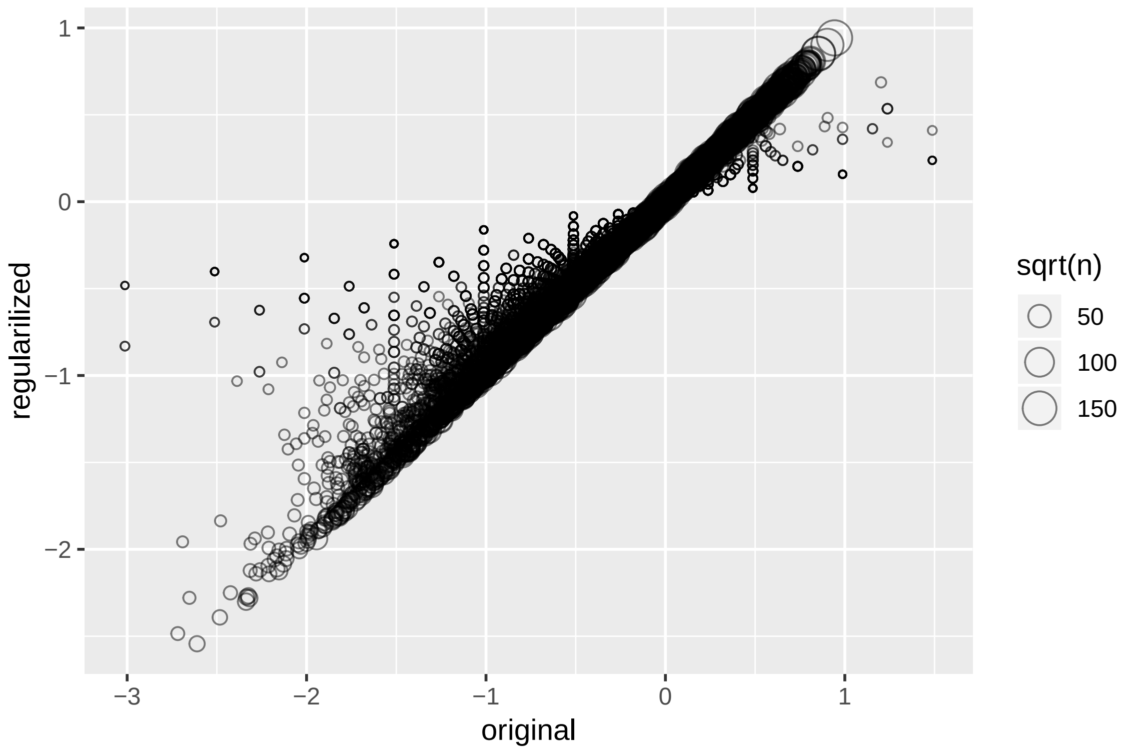
```
(lambda <- lambdas[which.min(rmses)])
```

```
## [1] 5.25
```

To see the effect that this had on the data, we will recompute the movie effect using our chosen lambda value, and then plot this against the original data:

```
movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_hat)/(n()+lambda), n_i = n())

# To see the effect, we plot the original movie data vs. our new averages:
data_frame(original = movie_avgs$b_i,
           regularilized = movie_reg_avgs$b_i,
           n = movie_reg_avgs$n_i) %>%
  ggplot(aes(original, regularilized, size=sqrt(n))) +
  geom_point(shape=1, alpha=0.5)
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```



This plot shows that there is a significant number of infrequently rated movies in the data that have large deviations from the mean (as described by the smallest circles). By reducing their effect on our calculations, the effect of the movies with larger number of ratings (larger circles) will be more strongly represented, and should be more representative of the majority of user sentiment.

We can now take another look at the residual sample to see how they have been affected by this. We are now seeing more familiar titles in our list:

```
validation %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  mutate(residual = rating - (mu_hat + b_i)) %>%
  arrange(desc(abs(residual))) %>%
  select(title,  residual) %>% distinct() %>% slice(1:15)
```

```
##                                                                        title
## 1                                           Shawshank Redemption, The (1994)
## 2                                                      Godfather, The (1972)
## 3                                                     PokÃ©mon Heroes (2003)
## 4                                                 Usual Suspects, The (1995)
## 5                                                    Schindler's List (1993)
## 6                                                          Casablanca (1942)
## 7                                                         Rear Window (1954)
## 8                                                      Third Man, The (1949)
## 9                                  Seven Samurai (Shichinin no samurai) (1954)
## 10                                            Godfather: Part II, The (1974)
## 11                                                    Dark Knight, The (2008)
## 12 Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)
## 13                                      One Flew Over the Cuckoo's Nest (1975)
## 14                             Lives of Others, The (Das Leben der Anderen) (2006)
## 15                                                           Yojimbo (1961)
##     residual
## 1  -3.954955
## 2  -3.915099
## 3   3.879153
## 4  -3.865647
## 5  -3.863301
## 6  -3.820046
## 7  -3.818119
## 8  -3.810014
## 9  -3.805941
## 10 -3.801624
## 11 -3.795321
## 12 -3.794946
## 13 -3.792946
## 14 -3.787393
## 15 -3.779107
```

We also see a big change in the representation of the highest and lowest b_i scored movies in the number of ratings given for each:

```
edx %>% count(movieId) %>%
  left_join(movie_reg_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10)
```

```
## Joining, by = "movieId"
```

```
## # A tibble: 10 x 3
##    title                                             b_i     n
##    <chr>                                           <dbl> <int>
##  1 Shawshank Redemption, The (1994)                0.942 28015
##  2 Godfather, The (1972)                           0.903 17747
##  3 Usual Suspects, The (1995)                      0.853 21648
##  4 Schindler's List (1993)                         0.851 23193
##  5 Casablanca (1942)                               0.808 11232
##  6 Rear Window (1954)                              0.806  7935
##  7 Sunset Blvd. (a.k.a. Sunset Boulevard) (1950) 0.802  2922
##  8 Third Man, The (1949)                           0.798  2967
##  9 Double Indemnity (1944)                         0.796  2154
## 10 Paths of Glory (1957)                           0.794  1571
```

```
edx %>% count(movieId) %>%
  left_join(movie_reg_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i, n) %>%
  slice(1:10)
```

```
## Joining, by = "movieId"
## # A tibble: 10 x 3
##    title                                                   b_i     n
##    <chr>                                                 <dbl> <int>
##  1 From Justin to Kelly (2003)                           -2.54   199
##  2 SuperBabies: Baby Geniuses 2 (2004)                   -2.48    56
##  3 PokÃ©mon Heroes (2003)                                -2.39   137
##  4 Glitter (2001)                                        -2.30   339
##  5 Gigli (2003)                                          -2.28   313
##  6 Disaster Movie (2008)                                 -2.28    32
##  7 Pokemon 4 Ever (a.k.a. PokÃ©mon 4: The Movie) (2002) -2.28   202
##  8 Barney's Great Adventure (1998)                       -2.27   208
##  9 Carnosaur 3: Primal Species (1996)                    -2.25    68
## 10 Son of the Mask (2005)                                -2.14   165
```

By filtering out the obscure data, we now have a sample set that is more closely aligned with group opinion, and we have accounted for the effects from movie-to-moview variation as well as user-to-user differences. The result of all of these calculations is our final RMSE of:

```
(min(rmses))
```

```
## [1] 0.864817
```

Results

Let's put the results of our calculations into a table, so that we can compare them:

```
rmse_results <- data_frame(method = "Baseline (mean)", RMSE = baseline_rmse)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = movie_rmse))
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = user_rmse))

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User Effect Model",
                                     RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Baseline (mean)	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488
Regularized Movie + User Effect Model	0.8648170

Discussion of Results

Our model has now been constructed, and to recap is composed of the following:

1. A baseline mean of all ratings in the database;

2. A calculation of the effect of each movie;
3. A calculation of the effect from each user providing ratings;

4. A reweighting of the contribution to the above effects based upon the number of ratings given for each movie - the further from the baseline the rating is and the fewer the ratings, the less it contributes;

5. A series of calculations to determine the optimal value for the penalty term for the reweighting calculation above.

These terms are then summed into our model equation as follows:

Prediction = baseline + movie effect(movie) + user effect(user)

or

Rating = baseline + movie effect(movie) + user effect(user) + error

We use the root-mean-squared error term to estimate the quality of our model. Using the techniques outlined above, we have reduced the error term from 1.06 using just the baseline mean to 0.0864.

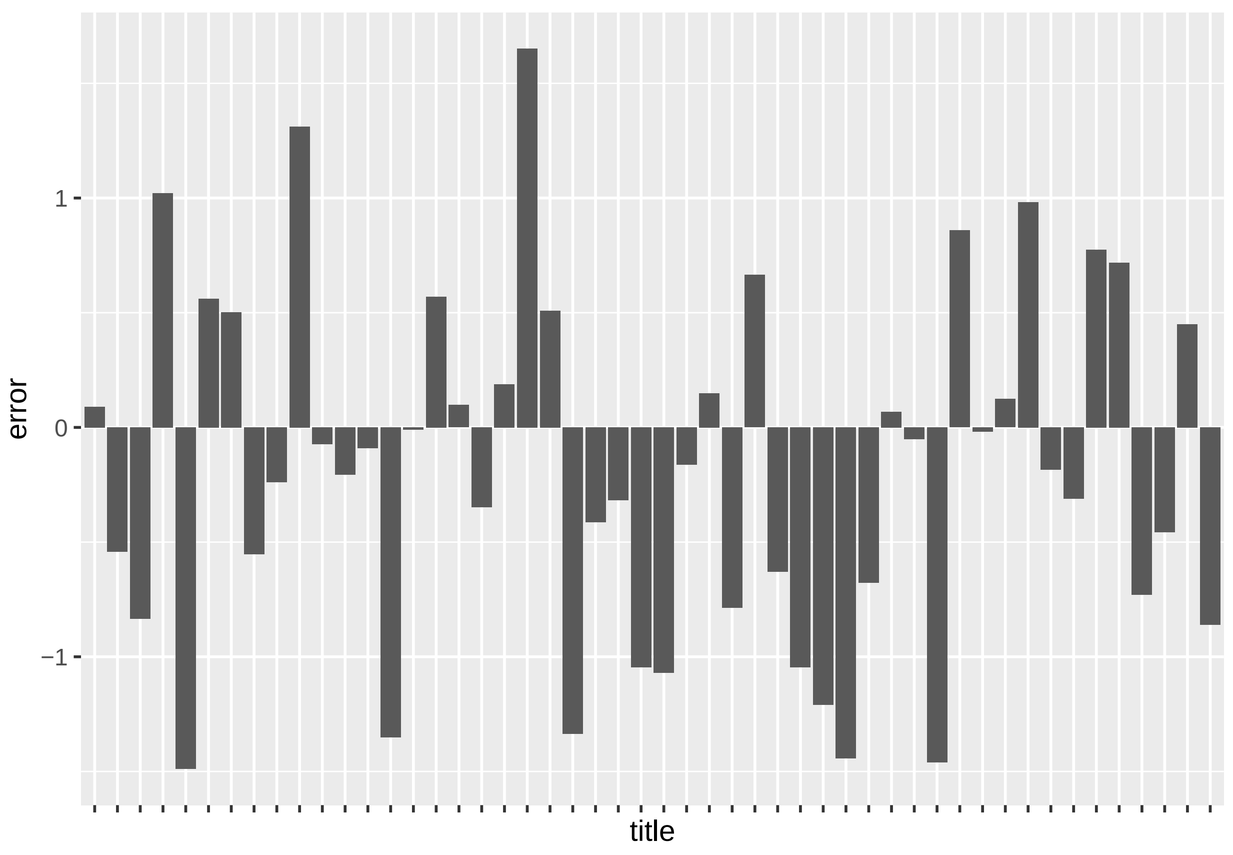
Now we will examine our results in more detail.

We first will take a sample of predictions from our validation set, and plot the error from each sample. We see that there are still significant errors in some of the individual samples, even though the group RMSE has been improved.

```
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
index <- sample(1:nrow(predicted_ratings_l), 50)

movie_sample <- predicted_ratings_l[index, ] %>% mutate(error = rating - pred, mu = mu_hat)

movie_plot <- gather(movie_sample, key = effect, value = value, b_i_l:mu, -pred)



movie_sample %>% ggplot(aes(x = title, y = error)) + geom_bar( stat = "identity") + theme(axis.text.x =
```



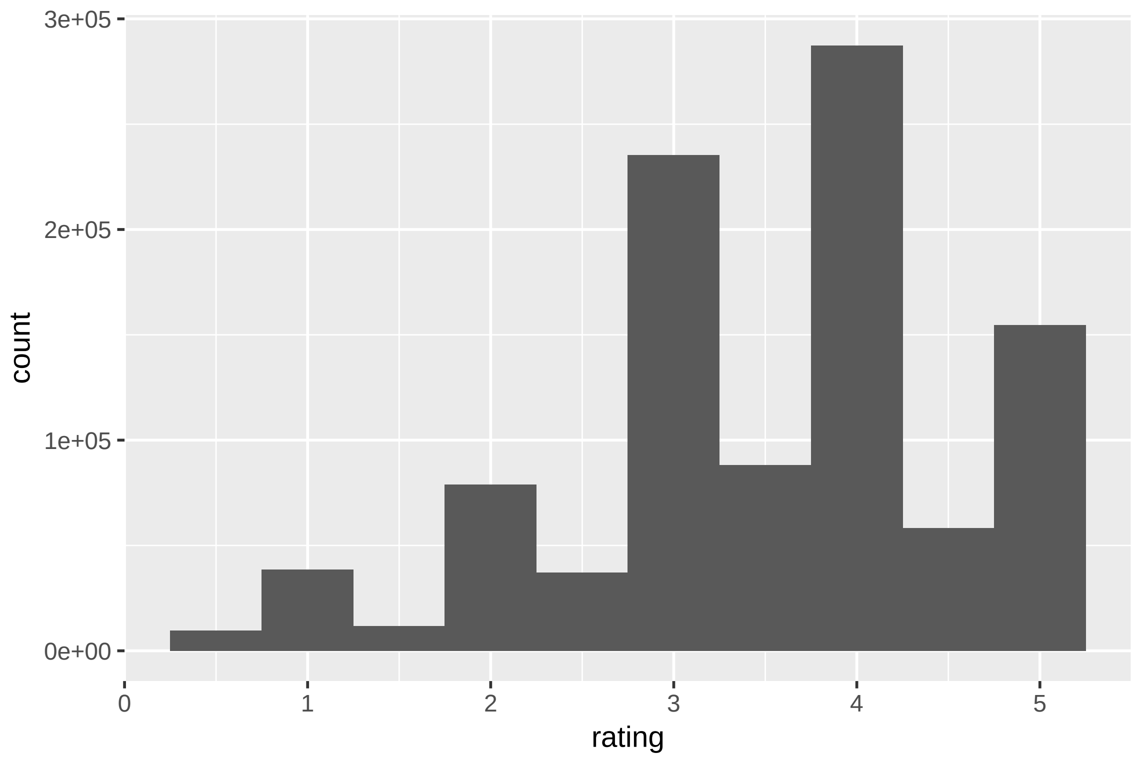
When we compare the distribution of ratings vs. predictions as shown in the summarize calculation, we see that they agree fairly closely. So the overall shape of the distribution of our model looks accurate, but there must be some more subtle effects present in the data.

```r
summary(predicted_ratings_l)
```

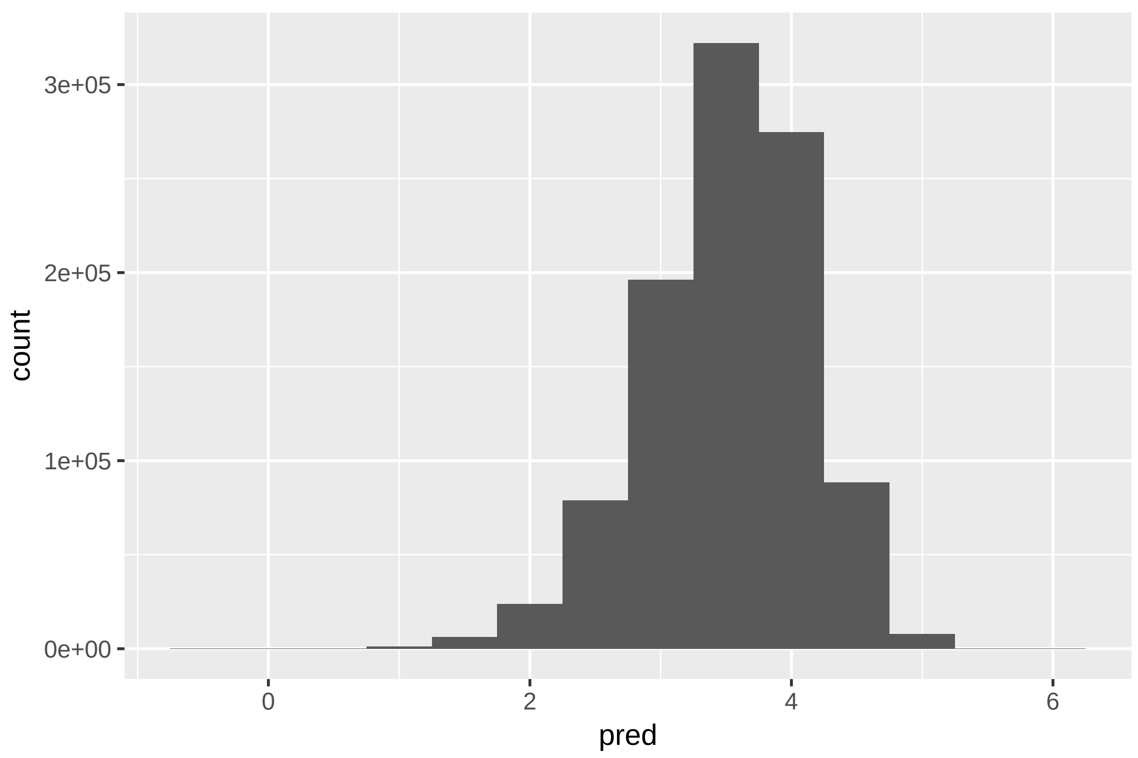
```
##      userId         movieId          rating        timestamp        
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08  
##  1st Qu.:18096   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.467e+08  
##  Median :35768   Median : 1827   Median :4.000   Median :1.035e+09  
##  Mean   :35870   Mean   : 4108   Mean   :3.512   Mean   :1.033e+09  
##  3rd Qu.:53621   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09  
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09  
##     title              genres              b_i_l          
##  Length:999999      Length:999999      Min.   :-2.543356  
##  Class :character   Class :character   1st Qu.:-0.293397  
##  Mode  :character   Mode  :character   Median : 0.078248  
##                                        Mean   : 0.001303  
##                                        3rd Qu.: 0.362842  
##                                        Max.   : 0.942489  
##      b_u_l               pred        
##  Min.   :-2.689712   Min.   :-0.4065  
##  1st Qu.:-0.227719   1st Qu.: 3.1392  
##  Median : 0.009002   Median : 3.5626  
##  Mean   :-0.004318   Mean   : 3.5095  
##  3rd Qu.: 0.236181   3rd Qu.: 3.9359
##  Max.   : 1.666833   Max.   : 5.9909
```

Another way to look at our data is to compare histograms of the actual ratings and the model ratings. Here we see that our model does not accurately capture the fact that most users select whole number ratings, rather than half-integer ratings.

```
predicted_ratings_l %>% ggplot(aes(x = rating)) +
  geom_histogram(binwidth = .5)
```



```
predicted_ratings_l %>% ggplot(aes(x = pred)) +
  geom_histogram(binwidth = .5)
```



While we have improved the prediction capability of our model over that of a naive baseline mean calculation, it is still possible to see errors in individual predictions of 1 star or more. Due to a limited amount of information available on individual user preferences (essentially extracted from the sparse ratings themselves), it may not be possible to reduce this error by a significant amount. As is discusssed in the next section, there are more sophisticated modeling techniques available that may help to better isolate specific user responses.

Areas for future investigation

From a review of the literature on recommendation systems, other factors that could be investigated for improvements to our model could include:

- Adjusting for when the movie was rated by utilizing the timestamp information. There is some evidence described in the literature that user preferences evolve over time, such as becoming more critical of movies the more movies a reviewer watches.
- Utilizing matrix factorization to assess even finer levels of correlations between details of the movies (genres, actors, directors) as well as finding correlations between groups of users as well.

Another area that bears further investigation is based on exploring the recommenderlab package which has support for collaborative filtering techniques for finding similarity patterns based on previous history and other factors.