HarvardX MovieLens Capstone Project

David Lovejoy

1/9/2020

Executive Summary

The aim of this analysis is to create a recommendation model, inspired by and measured against the winning team in the Netflix challenge. Since Netflix does not allow public access to their data, the data used in this analysis will be a subset of similar rating data provided by the GroupLens research lab, which can be found here:

https://grouplens.org/datasets/movielens/10m/

A recommendation model attempts to predict the rating a user would give an item based on how that user has responded to previous items or based on how other users with similar tastes — as expressed through their mutual ratings — have responded to the item. Large e-commerce companies like Amazon can generate massive datasets from their customer reviews and use this information to recommend further products to their customers, creating a virtuous cycle.

A large media-services provider, such as Netflix, can also benefit massively from using a highly accurate recommendation system, as it will at once provide a value to its customers who may not find the recommended titles otherwise, while at the same time ensure the likelihood they return to watch more and rate more. This aforementioned cycle can greatly help in making devoted, happy customers, and using them to attract yet more customers. Netflix uses a five-star rating system, with one star being the lowest rating a user can give, and five being the highest.

The Netflix challenge offered a prize of 1 million dollars to anyone who could improve the company's recommendation model by at least ten percent. Competitor submissions were ranked by their root mean squared error, or RMSE, on an undisclosed test set. While the final scores were never disclosed by Netflix, the goal of this analysis will be to achieve an RMSE score <= 0.8649.

Analysis

Importing and Partitioning Data

The MovieLens data provided by the GroupLens research lab consists of 10 million ratings applied to 10 thousand movies by 72 thousand users. After it is imported, the data will be partitioned into a train set and a validation set, the latter remaining untouched and used to test the accuracy of the predictive model.

We begin by installing any missing packages, downloading the dataset, and partitioning it into two sets: edx and validation, or train and test, respectively.

```
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")

# 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)` instead
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)
```

Data Exploration

To get a sense of how the data is structured:

```
edx %>% as_tibble()
```

```
## # A tibble: 9,000,055 x 6
##    userId movieId rating timestamp title               genres
##     <int>   <dbl>  <dbl>     <int> <chr>               <chr>
##  1      1     122      5 838985046 Boomerang (1992)    Comedy|Romance
##  2      1     185      5 838983525 Net, The (1995)     Action|Crime|Thrill~
##  3      1     292      5 838983421 Outbreak (1995)     Action|Drama|Sci-Fi~
##  4      1     316      5 838983392 Stargate (1994)     Action|Adventure|Sc~
##  5      1     329      5 838983392 Star Trek: Generat~ Action|Adventure|Dr~
##  6      1     355      5 838984474 Flintstones, The (~ Children|Comedy|Fan~
##  7      1     356      5 838983653 Forrest Gump (1994) Comedy|Drama|Romanc~
##  8      1     362      5 838984885 Jungle Book, The (~ Adventure|Children|~
##  9      1     364      5 838983707 Lion King, The (19~ Adventure|Animation~
## 10      1     370      5 838984596 Naked Gun 33 1/3: ~ Action|Comedy
## # ... with 9,000,045 more rows
```

The table is in tidy format and has thousands of rows, in this case user ratings. To find out how many unique users and movies there are:

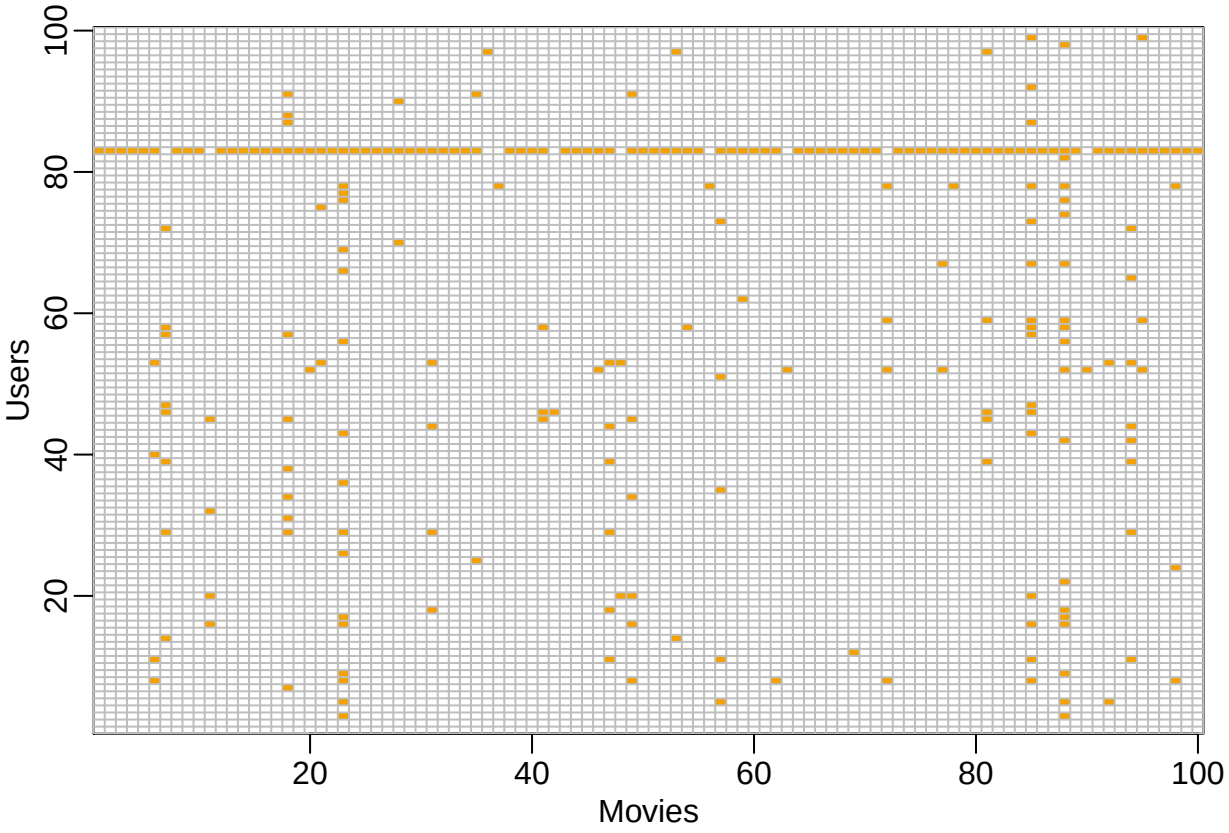
```
edx %>%
summarize(n_users = n_distinct(userId),
n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

The product of these two numbers suggest there should be many more ratings than there are rows. The enormous discrepancy implies not all users rated all movies, quite the opposite in fact.

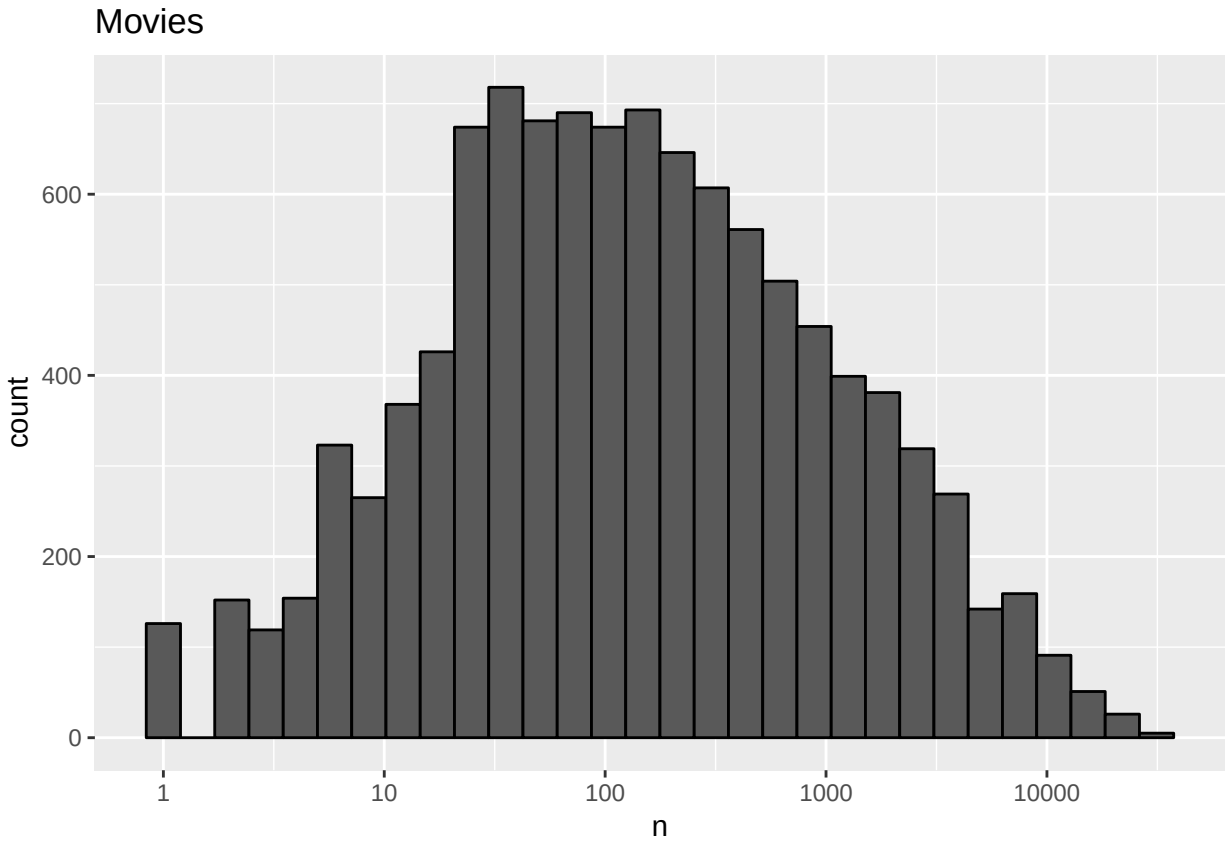
Visualization

This is perhaps best visualized with a sparse matrix, but since the dataset is quite large, a random sample of 100 users and 100 movies can show the following:

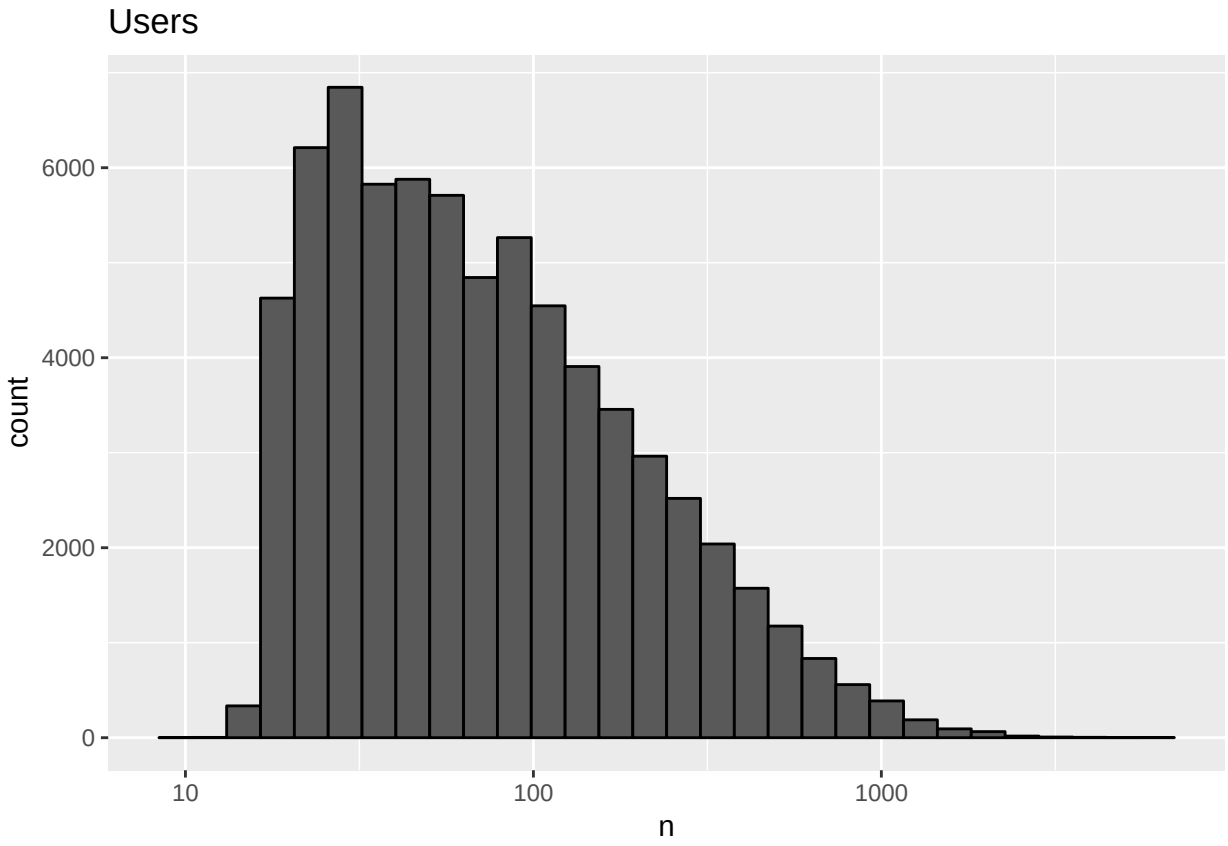


The yellow squares denote user-movie combinations for which ratings were received. The role of the recommendation system is to fill in the missing values with predictions. Even with the vast majority of ratings missing, there is actually more data to draw from than may readily be apparent. To predict how a particular user would rate a movie they have not yet seen, predictors can be made from how that user has rated other titles and how other users have rated the movie in question. Essentially, each predicted rating will be using data from the entire matrix.

Looking at the distribution of movie ratings, it is clear some movies are rated more than others. This is not unexpected when considering cinema's history, film distribution and box office performances have been anything but even. It stands to reason then, that theme, production value, and crew popularity would influence how many ratings a certain movie receives.



Another factor to consider, is that some users leave more ratings than others, either because they watch more movies, or because they feel compelled to for one reason or another.



Evaluation

A loss function is used to measure the performance of various machine learning models against each other or some baseline. The one used by Netflix was the root mean squared error (RMSE), where a value greater than 1 denotes a predicted rating that is off by more than one star, which is not very accurate.

If we define $y_{u,i}$ as the rating for movie i by user u and our prediction as $\hat{y}_{u,i}$, the RMSE is defined as:

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

N is the number of user/movie combinations and the sum is occurring over all these combinations.

Here is a function that computes the RMSE for vectors of ratings and their corresponding predictors:

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

Naïve Model

This model-based approach is the simplest possible recommendation system, in which the same rating is predicted for all movies regardless of user:

$$Y_{u,i} = \mu + \epsilon_{u,i}$$

with $\epsilon_{u,i}$ representing independent errors sampled from the same distribution centered at 0, and μ representing the "true" rating for all movies and users. The estimate that minimizes the RMSE is the least squares estimate of μ. In this case, it is just the average of all ratings in the training set, which is:

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

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

Predicting all unknown ratings in the validation set with $\hat{\mu}$ yields the following RMSE:

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

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

Plugging in any other number will result in a higher RMSE because the average minimizes the RMSE in this model. Netflix expected better results, and to win the million-dollar grand prize, participants had to achieve an RMSE of approximately 0.857.

It will be helpful to create a table to compare the results achieved with each model:

```
rmse_results <- data.frame(method = "Naive Model", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Model	1.061202

Movie Effect Model

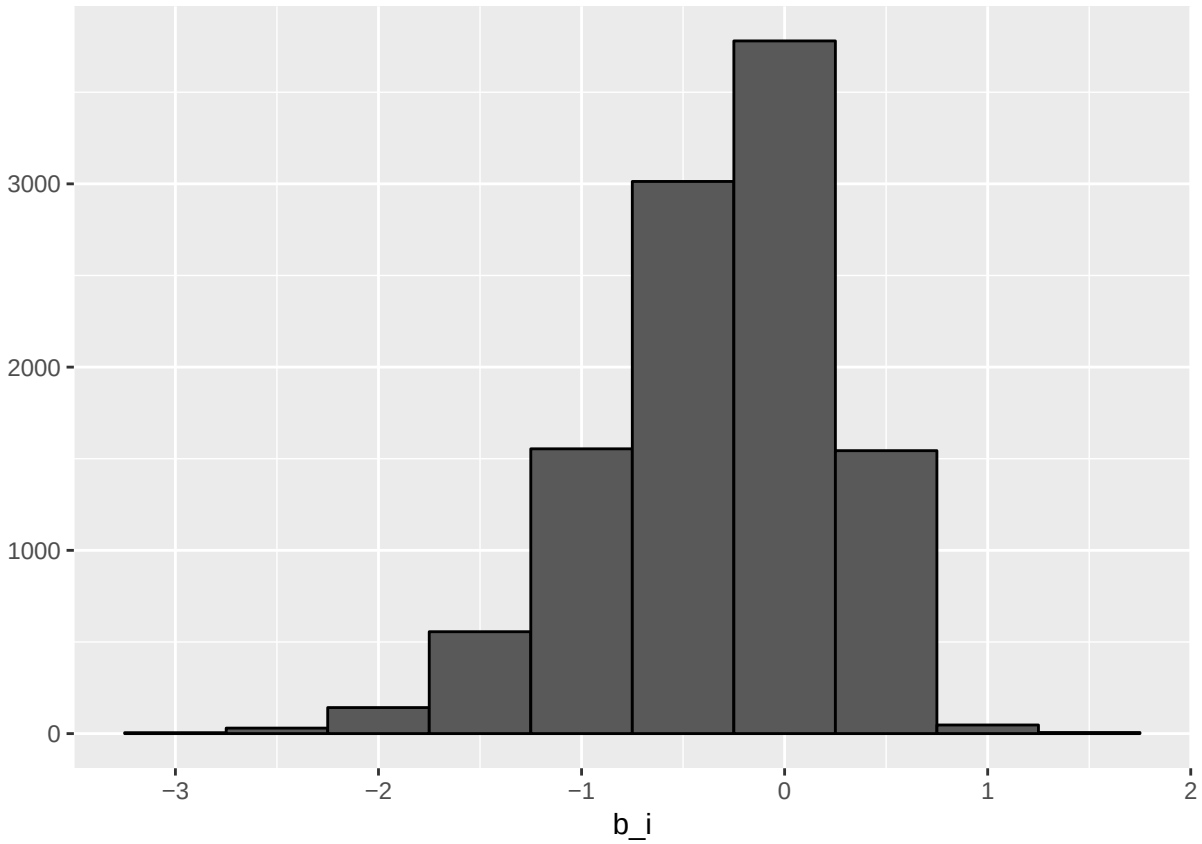
It is prudent to account for potential bias in the data. In the case of movie ratings, the data confirms some movies receive far more ratings than others. Adding the term b_i to represent the average ranking for movie i adjusts the previous model to account for this:

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

The b notation represents bias, and the least square estimate b_i is just the average of $Y_{u,i} - \hat{\mu}$ for each movie i. Which is computed as follows:

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

There is substantial variation among these estimates, which makes sense considering some movies are subjectively, if not objectively better.



$\hat{\mu} = 3.5$ so a $b_i = 1.5$ is a perfect five star rating.

Using $\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i$ as the next model:

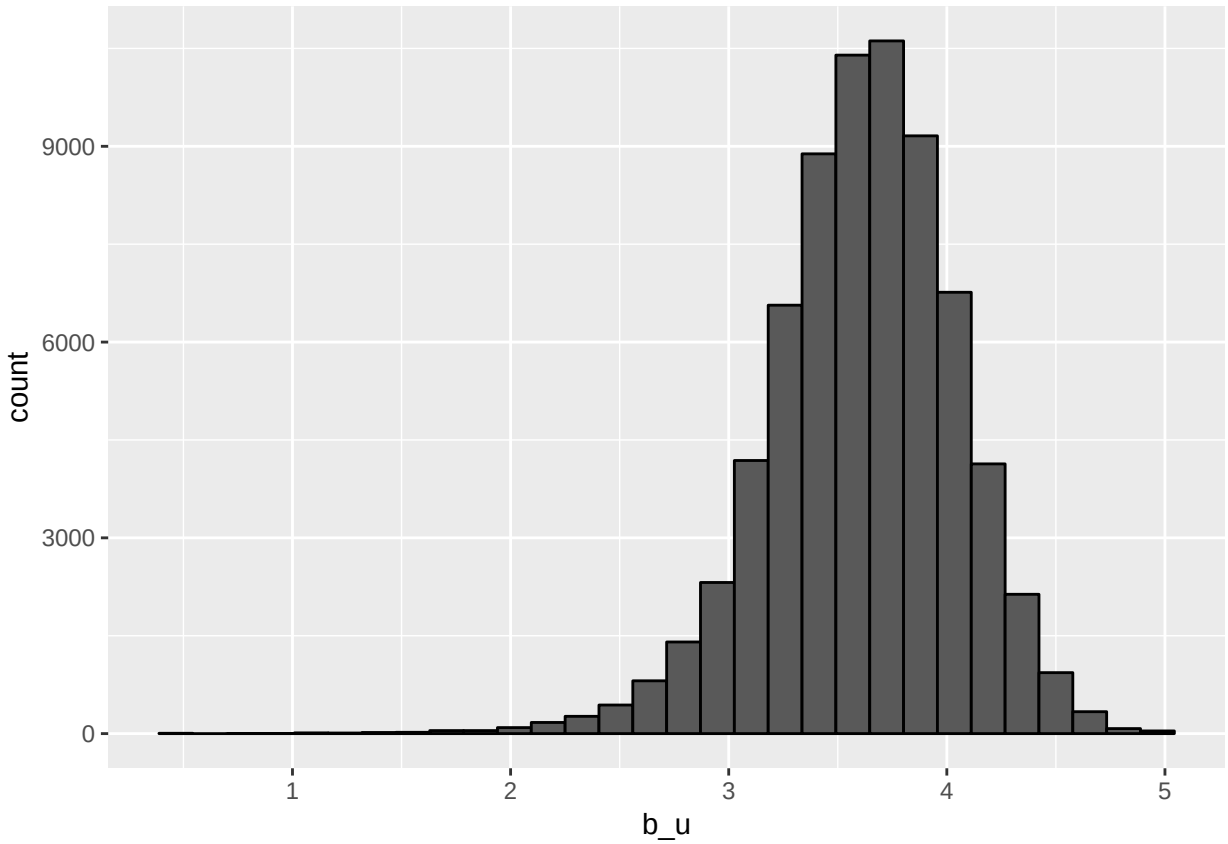
```
predicted_ratings <- mu + validation %>%
left_join(movie_avgs, by='movieId') %>%
.$b_i

model_1_rmse <- RMSE(validation$rating, predicted_ratings)
rmse_results <- bind_rows(rmse_results,
data_frame(method="Movie Effect Model",
RMSE = model_1_rmse))

rmse_results %>% knitr::kable()
```

method	RMSE
Naive Model	1.0612018
Movie Effect Model	0.9439087

There is about an 11% improvement when compared to the Naïve model.

Movie + User Effect Model

Computing the average rating for user u for those that have rated over 100 movies to look for variability across user data reveals the following:



Some users are hard-to-please, and others cannot be disappointed, implying a further improvement to the model may be:

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

with b_u denoting a user-specific effect. Now if a hard-to-please user (negative b_u) rates a great movie (positive b_i), the effects counter each other and the model may be able to correctly predict that this user rated a five-star movie with three stars.

Computing an approximation with $\hat{\mu}$ and $\hat{b_i}$ and estimating $\hat{b_u}$ as the average of $y_{u,i} - \hat{\mu} - \hat{b_i}$:

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

This allows for more accurate predictors to be constructed and improves the RMSE:

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

model_2_rmse <- RMSE(validation$rating, predicted_ratings)
rmse_results <- bind_rows(rmse_results,
data_frame(method="Movie + User Effects Model",
```

```
RMSE = model_2_rmse))

rmse_results %>% knitr::kable()
```

method	RMSE
Naive Model	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488

Regularized Movie Effect Model

Large estimates that come from small sample sizes can add unwanted noise to the data, negatively affecting the accuracy of the predictive model as seen by looking at the 10 best movies and how many ratings they received:

```
##  [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] "Human Condition II, The (Ningen no joken II) (1959)"
##  [9] "Human Condition III, The (Ningen no joken III) (1961)"
## [10] "Constantine's Sword (2007)"

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

Looking at the 10 worst movies and their ratings further reveals this effect.

```
##  [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] "Hip Hop Witch, Da (2000)"
##  [8] "Disaster Movie (2008)"
##  [9] "From Justin to Kelly (2003)"
## [10] "Criminals (1996)"

##  [1]   2   1   1   1   2  56  14  32 199   2
```

It becomes clear that many of these are not movies seen, and certainly not rated, by the masses. The fewer the users, the greater the uncertainty. The concept of regularization penalizes these estimates, as expressed in this equation:

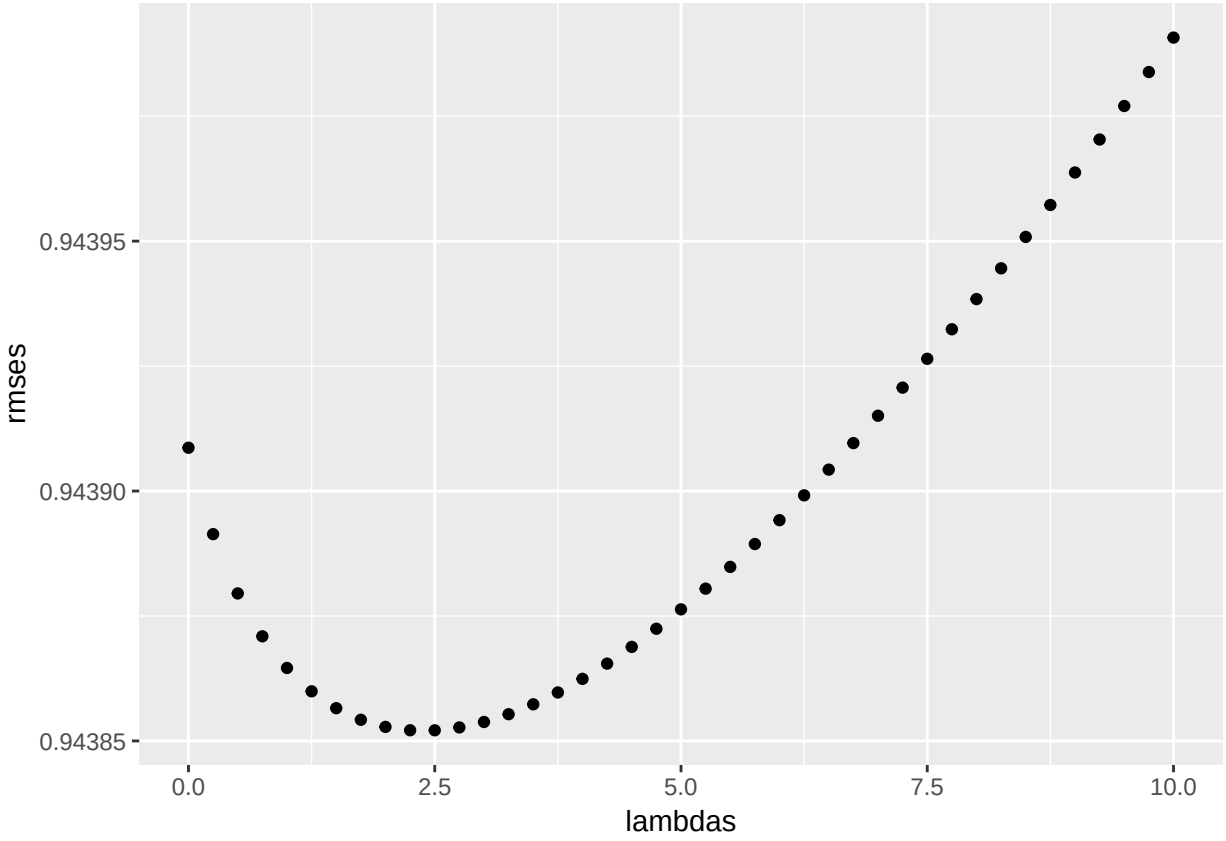
$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i\right)^2+\lambda\sum_i b_i^2$$

The second term is a penalty that gets larger as b_i gets larger. The values of b_i that minimize this equation are:

$$\hat{b}_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (Y_{u,i} - \hat{\mu})$$

n_i denotes the number of ratings made for movie i. When the sample size n_i is very large, then the penalty λ is effectively ignored since $n_i + \lambda \approx n_i$. However, when the n_i is small, then the estimate $\hat{b}_i(\lambda)$ is shrunken towards 0. The larger λ, the more the estimate shrinks.

As λ is a tuning parameter, it is important to find its optimal value using cross-validation.

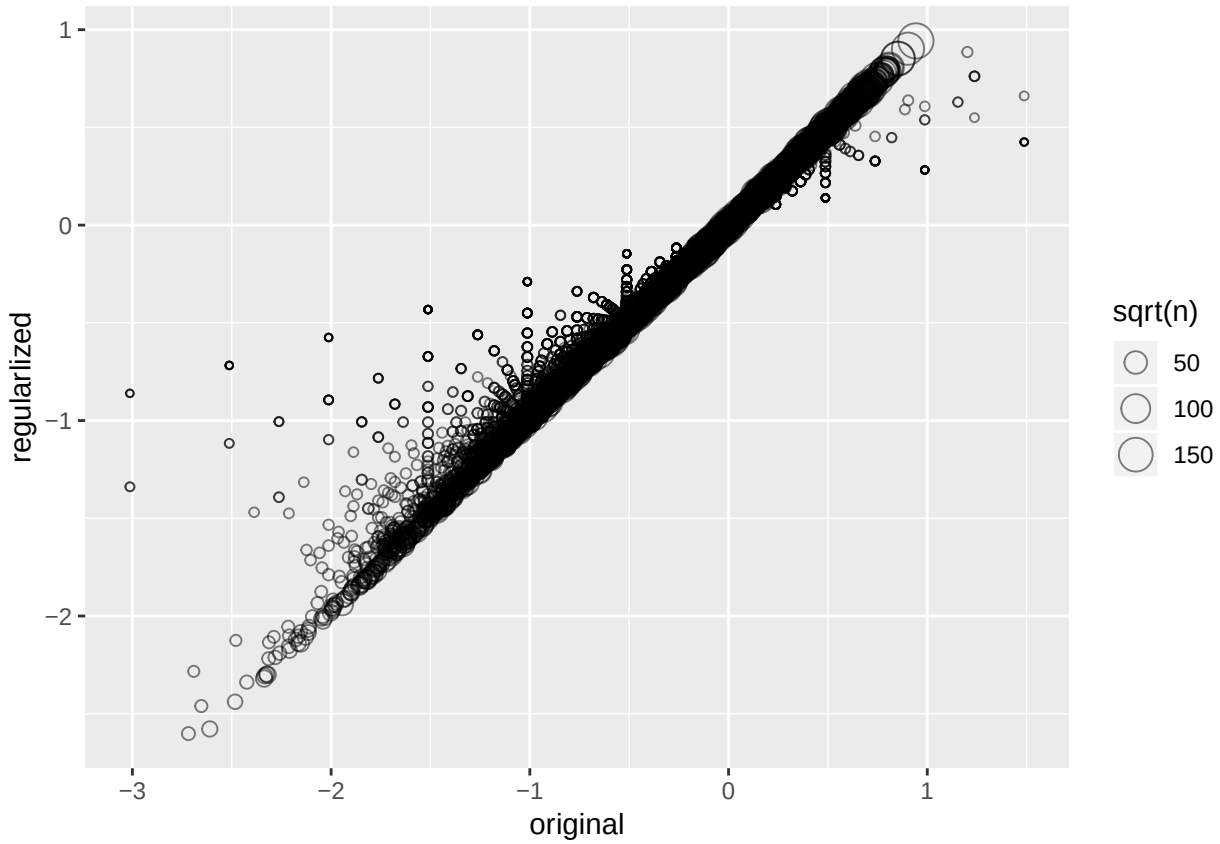



```
## [1] 2.5
```

Computing the regularized estimates of b_i using $\lambda = 2.5$

```
lambda <- 2.5
mu <- mean(edx$rating)
movie_reg_avgs <- edx %>%
group_by(movieId) %>%
summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
```

Plotting how the regularized estimates shrink compared to the least squares estimates of the previous model shows the following:



The top 10 best movies based on the penalized estimates $\hat{b}_i(\lambda)$ make a lot more sense now:

```
##  [1] "Shawshank Redemption, The (1994)"
##  [2] "Godfather, The (1972)"
##  [3] "More (1998)"
##  [4] "Usual Suspects, The (1995)"
##  [5] "Schindler's List (1993)"
##  [6] "Casablanca (1942)"
##  [7] "Rear Window (1954)"
##  [8] "Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)"
##  [9] "Third Man, The (1949)"
## [10] "Double Indemnity (1944)"
```

The same can be said for the 10 worst movies:

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

Testing this regularized movie effect model yields the following RMSE:

```r
predicted_ratings <- validation %>%
left_join(movie_reg_avgs, by = "movieId") %>%
mutate(pred = mu + b_i) %>%
.$pred

model_3_rmse <- RMSE(validation$rating, predicted_ratings)
rmse_results <- bind_rows(rmse_results, data_frame(method="Regularized Movie Effect Model",
RMSE = model_3_rmse))

rmse_results %>% knitr::kable()
```

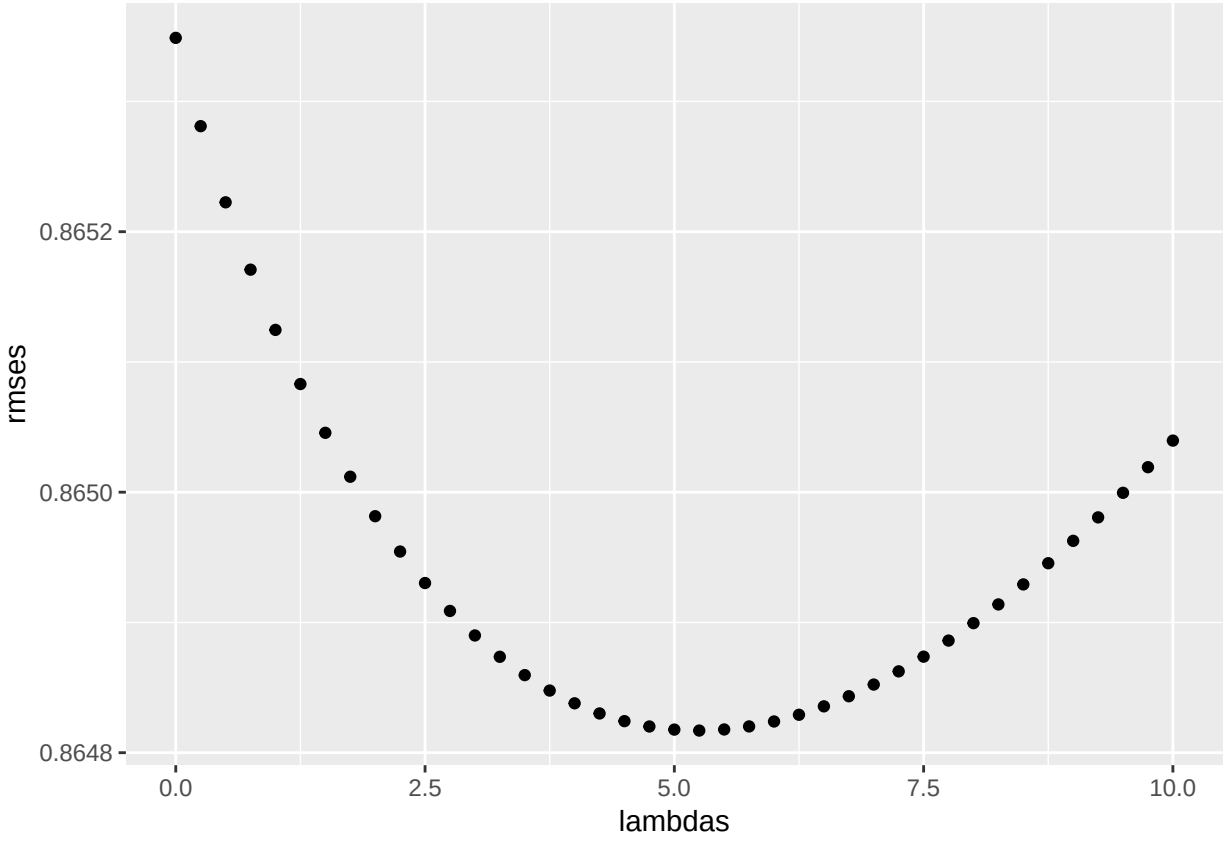
method	RMSE
Naive Model	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488
Regularized Movie Effect Model	0.9438521

Regularized Movie + User Effect Model

The following equation also regularizes the user effect:

$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu b_i b_u\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right)$

Using cross-validation to find the optimal lambda value for this model reveals the following:



To calculate the optimal λ for the full model:

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

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

The final RMSE achieved with this:

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

rmse_results %>% knitr::kable()
```

method	RMSE
Naive Model	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488
Regularized Movie Effect Model	0.9438521
Regularized Movie + User Effect Model	0.8648170

Results

Starting with the naïve approach defined the upper limit of RMSE values in this analysis, essentially one standard deviation from the true rating. As that model was not tuned any more than simply taking the average of all the ratings, there was certainly room for improvement. Accounting for movie bias in the Movie Effect Model yielded the largest change to the RMSE value; and addressing user bias further reduced the RMSE value in the Movie + User Effect Model by a significant 8%.

Finding bias in a dataset comprised of user ratings is decidedly less surprising than seeing the very similar RMSE results between the Regularized Movie Effect Model and the Movie Effect Model. This was unexpected, as the rating accuracy seemed to improve markedly using the penalized least squares approach. The best model, however, is the Regularized Movie + User Effect model, as seen here:

```
rmse_results %>% knitr::kable()
```

method	RMSE
Naive Model	1.0612018
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8653488
Regularized Movie Effect Model	0.9438521
Regularized Movie + User Effect Model	0.8648170

Conclusion

The final RMSE of **0.86481** was achieved using the Regularized Movie + User Effect Model, and while that accomplishes the goal set out in the beginning, there is still much room for optimization.

Using matrix factorization to account for hitherto ignored variation in the data might yield a better RMSE value. For example, there are similar rating patterns between groups of movies and groups of users, such as critically acclaimed movies versus blockbusters, or users who like gangster movies enough to override any review disparity between them. Augmenting the previous formula $Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$ by incorporating the vectors p and q which account for residual correlation in the data, should be able to explain much more of the variance in the data. The vectors in the new formula $Y_{u,i} = \mu + b_i + b_u + p_u q_i + \varepsilon_{u,i}$ do not accurately represent the structural complexity in the data, however, so using singular value decomposition (SVD) to perform principal component analysis (PCA), or finding the vectors needed to explain the variability of the residual matrix would be a good next step.