# MovieLens - Movie Recommendation System

Jayesh John

06/03/2020

## Introduction

In October 2006, Netflix offered a geeky and formidable challenge to the data science community: come up with a recommendation algorithm that could do a better job accurately predicting the movies customers would like than Netflix's in-house software, *Cinematch*. To qualify for the prize of a million dollar, entries had to be at least 10 percent better than *Cinematch*.

Netflix uses a recommendation system to predict how many stars a user will give a specific movie on a scale of 1-5. One star suggests it is not a good movie, whereas five stars suggests it is an excellent movie. Movies for which a high rating is predicted for a given user are then recommended to that user.

For this project, we will be creating a movie recommendation system using the MovieLens dataset. Recommendation systems are more complicated machine learning challenges because each outcome has a different set of predictors. For example, different users rate a different number of movies and rate different movies. The objective is to model an algorithm that predicts movie ratings with the least residual mean squared error (RMSE).

## The MovieLens Dataset

### Load Libraries and Dataset

The Netflix data is not publicly available, but the GroupLens research lab114 generated their own database with over 20 million ratings for over 27,000 movies by more than 138,000 users. The MovieLens 10M dataset is a subset dataset that is being used for this project:

```
################################
# Create edx set, validation set
################################

# Note: this process could take a couple of minutes

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")
if(!require(dplyr)) install.packages("dplyr", repos = "http://cran.us.r-project.org")

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip
# https://drive.google.com/drive/folders/1IZcBBXOOmL9wu9AdzMBFUG8GoPbGQ38D?usp=sharing
```

## Create Train and Validation Sets

The following code is used to generate *Train* and *Validation* sets in the ratio 90:10. The Models are trained using the *Train* set. The prediction accuracy of the models are further tested using the *Validation* set and RMSE determined.

```
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,]
```

```
# Alternatively the train and validation data can be downloaded from the below google drive link
# https://drive.google.com/drive/folders/1IZcBBXOOmL9wu9AdzMBFUG8GoPbGQ38D?usp=sharing
edx <- readRDS("~/projects/movielens/edx.rds")
temp <- readRDS("~/projects/movielens/validation.rds")

# 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(temp, removed)
```

## Tidy View of Data

MovieLens dataset has 10 million rows with each row representing a rating given by one user to one movie. The dataset has 6 columns or features. The dataset is partitioned into *train* and *validation* sets containing ~9M and ~1M ratings respectively.

We can see the edx *train* table is in tidy format with millions of rows:

```
# see 'edx' train data in tidy format
edx %>% 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|Thriller
##  3      1     292      5 838983421 Outbreak (1995)        Action|Drama|Sci-Fi|T~
##  4      1     316      5 838983392 Stargate (1994)        Action|Adventure|Sci-~
##  5      1     329      5 838983392 Star Trek: Generation~ Action|Adventure|Dram~
##  6      1     355      5 838984474 Flintstones, The (199~ Children|Comedy|Fanta~
##  7      1     356      5 838983653 Forrest Gump (1994)    Comedy|Drama|Romance|~
##  8      1     362      5 838984885 Jungle Book, The (199~ Adventure|Children|Ro~
##  9      1     364      5 838983707 Lion King, The (1994)  Adventure|Animation|C~
## 10      1     370      5 838984596 Naked Gun 33 1/3: The~ Action|Comedy
## # ... with 9,000,045 more rows
```

## Data Summary

Each row represents a rating given by one user to one movie. We can summarize the number of unique users that provided ratings and how many unique movies were rated in the *train* set:

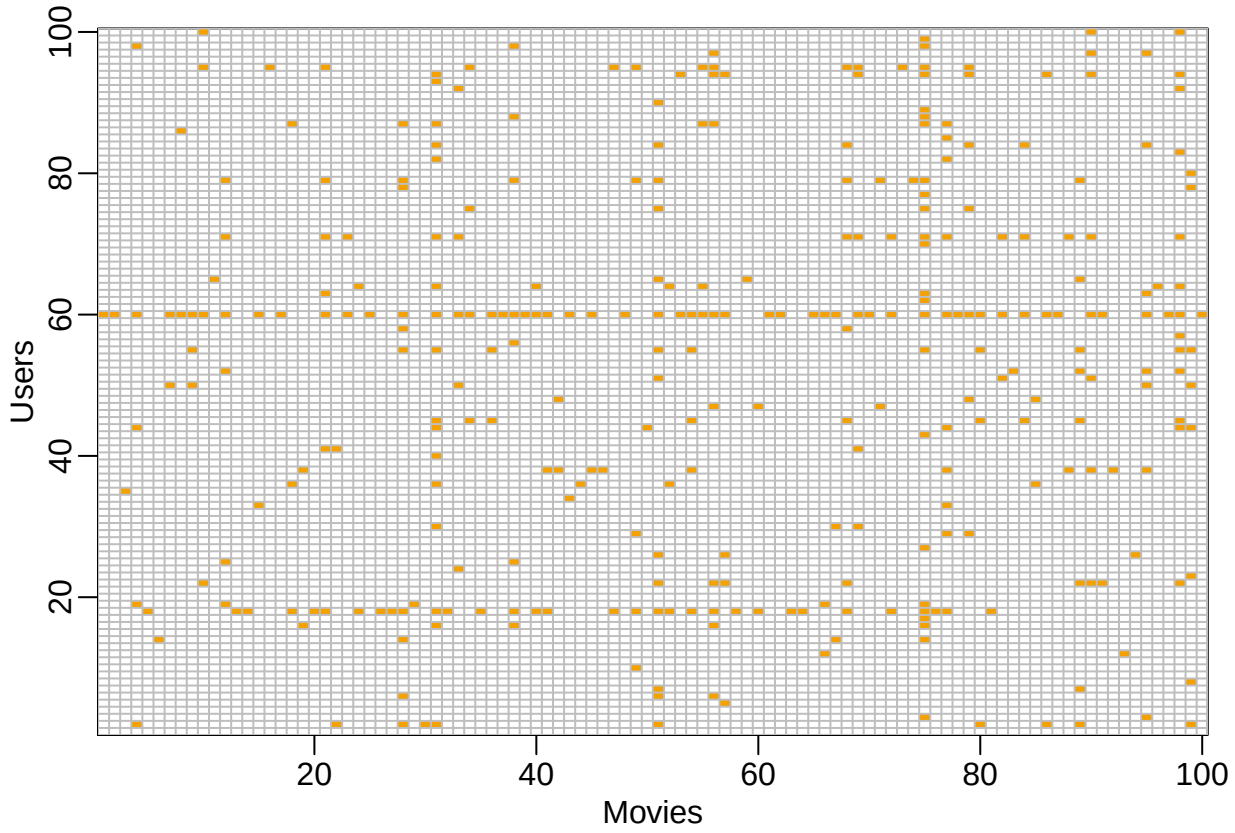
```
# number of unique users and movies
edx %>% summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

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

If we multiply those two numbers we get a number much larger than 10M which means that not all users rated all movies. We can therefore think of this data as a sparse matrix with users on the rows and movies on the columns, with many empty cells.

You can think of the task of a recommendation system as filling in those empty values. To see how sparse the matrix is, here is the matrix for a random sample of 100 movies and 100 users with yellow indicating a user/movie combination for which we have a rating.

```
# show sparse matrix for sampled 100 unique userId and moveiId
users <- sample(unique(edx$userId), 100)
rafalib::mypar()
edx %>% filter(userId %in% users) %>%
   select(userId, movieId, rating) %>%
   mutate(rating = 3) %>%
   spread(movieId, rating) %>% select(sample(ncol(.), 100)) %>%
   as.matrix() %>% t(.) %>%
   image(1:100, 1:100,. , xlab="Movies", ylab="Users")
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
```



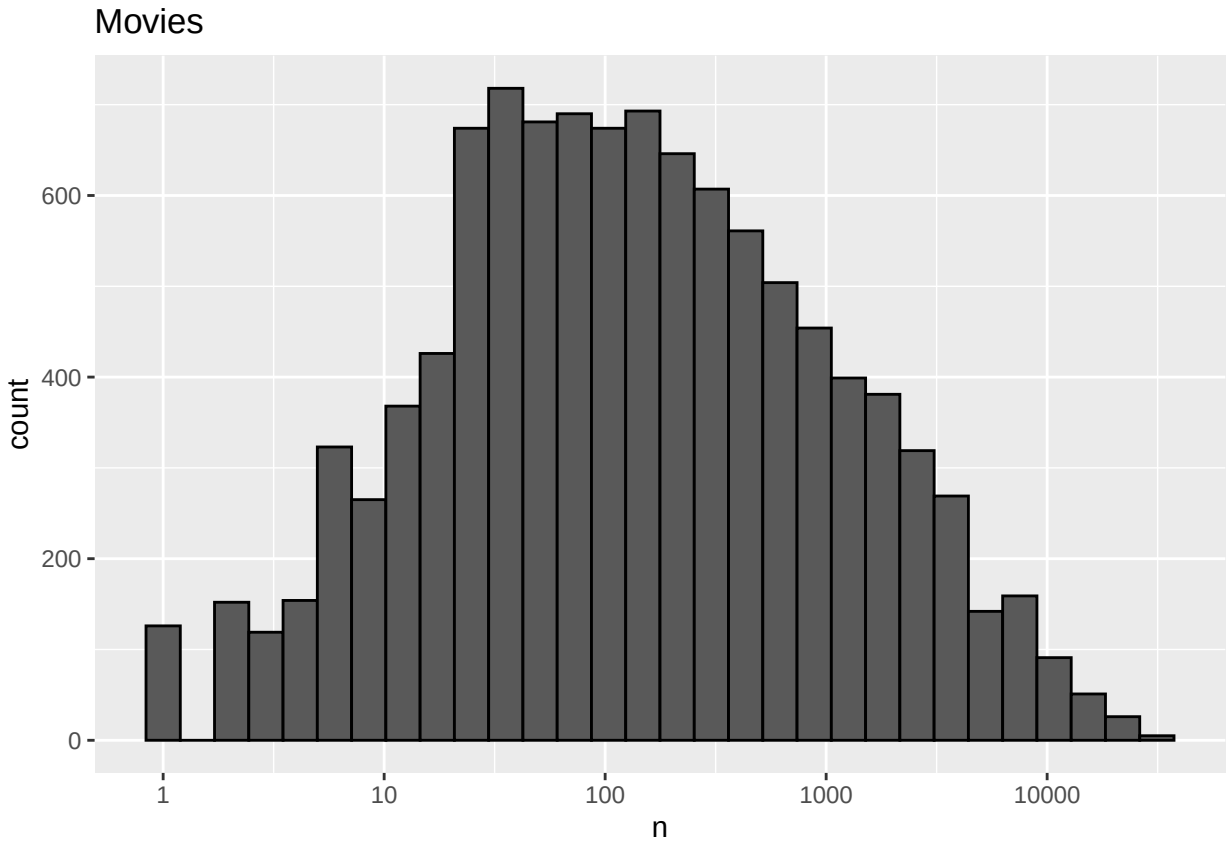

Note that if we are predicting the rating for movie $i$ by user $u$ , in principle, all other ratings related to movie $i$ and by user $u$ may used as predictors, but different users rate different movies and a different number of movies. Furthermore, we may be able to use information from other movies that we have determined are similar to movie $i$ or from users determined to be similar to user $u$ . In essence, the entire matrix can be used as predictors for each cell.

## Exploratory Data Analysis

Let's look at some of the general properties of the data to better understand the challenges.

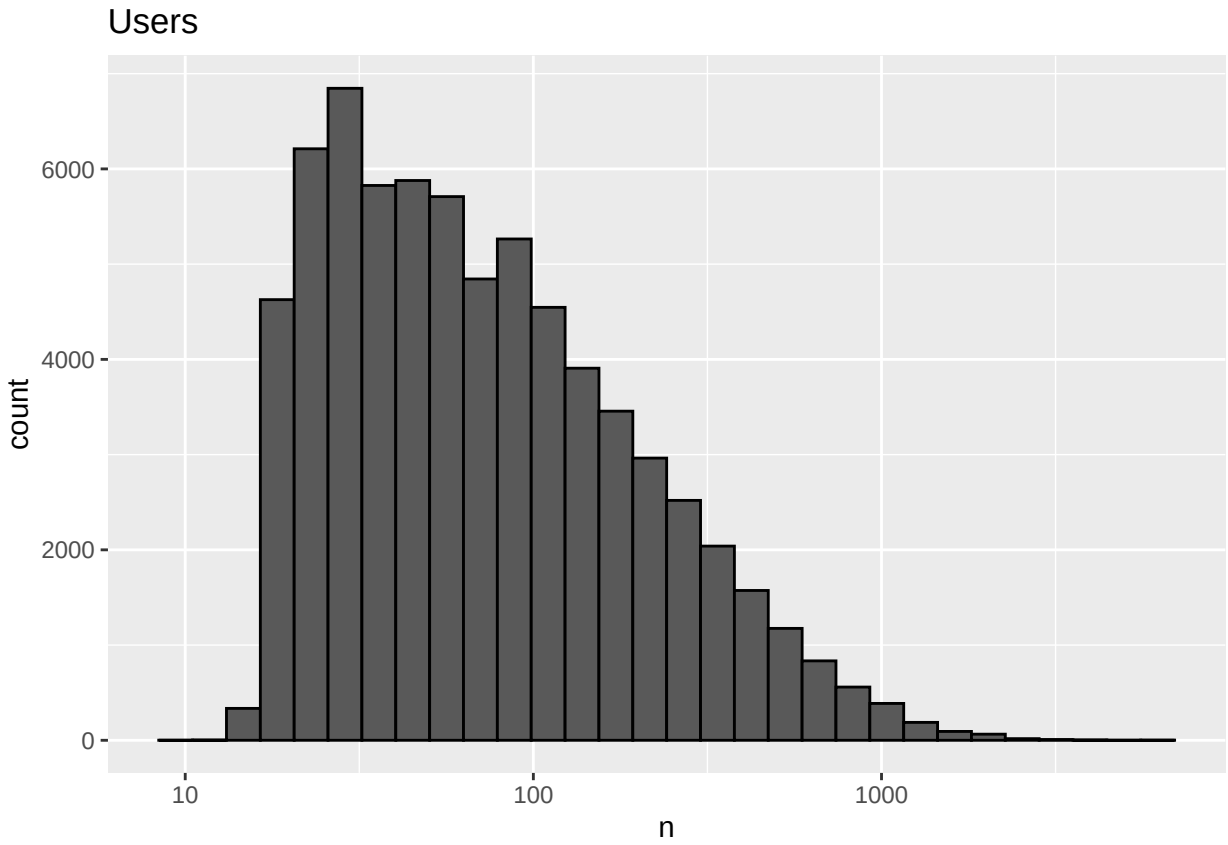
*First Observation* - We notice that some movies get rated more than others. This should not surprise us given that there are blockbuster movies watched by millions and artsy, independent movies watched by just a few. Below is the distribution:

```
# distribution of movie ratings
edx %>%
   dplyr::count(movieId) %>%
   ggplot(aes(n)) +
   geom_histogram(bins = 30, color = "black") +
   scale_x_log10() +
   ggtitle("Movies")
```




*Second Observation* - Some users are more active than others at rating movies. Below is the distribution:

```
# distribution of users
edx %>%
   dplyr::count(userId) %>%
   ggplot(aes(n)) +
   geom_histogram(bins = 30, color = "black") +
   scale_x_log10() +
   ggtitle("Users")
```




# Machine Learning Data Modeling

## RMSE - Loss Function

The typical error loss *residual mean squared error* (RMSE) on the test set is used as the metric to determine the performance of our models. If we define $y_{u,i}$ as the rating for movie $i$ by user $u$ and denote out prediction with $\hat{y}_{u,i}$, then RMSE is defined as:

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

with $N$ being the number of user/movie combinations and the sum occurring over all these combinations.

The RMSE can be interpreted similar to standard deviation: it is the typical error we make when predicting a movie rating. If this number is larger than 1, it means our typical error is larger than one star, which is not good.

Let's write a function that computes the RMSE for vectors of ratings and their corresponding predictors:

```
# RMSE function for vectors of ratings and their corresponding predictors
RMSE <- function(true_ratings, predicted_ratings){
   sqrt(mean((true_ratings - predicted_ratings)^2))
 }
```

## First model: Single Value Mean

Let's start by building the simplest possible recommendation system: we predict the same rating for all movies regardless of user. What number should this prediction be? We can use a model based approach to answer this. A model that assumes the same rating for all movies and users with all the differences explained by random variation would look like this:

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

with $\epsilon_{u,i}$ independent errors sampled from the same distribution centered at 0 and $\mu$ the "true" rating for all movies. We know that the estimate that minimizes the RMSE is the least squares estimate of $\mu$ and, in this case, is the average of all ratings:

```
# Model 1: predict same rating for all movies using average of all ratings
# predict average rating of all movies
mu_hat <- mean(edx$rating)
mu_hat
```

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

If we predict all unknown ratings with $\hat{\mu}$ we obtain the following RMSE:

```
# calculate rmse of this naive approach
naive_rmse <- RMSE(validation$rating, mu_hat)
naive_rmse
```

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

As we go along, we will be comparing different approaches. Let's start by creating a results table with this naive approach:

```
# add rmse results in a table
rmse_results <- data.frame(method = "Single Value Mean", RMSE = naive_rmse)
 rmse_results %>% knitr::kable()
```

| method | RMSE |
|---|---|
| Single Value Mean | 1.061202 |

## Second model: Modeling Movie Effects

We know from experience that some movies are just generally rated higher than others. This intuition, that different movies are rated differently, is confirmed by data. We can augment our previous model by adding the term $b_i$ to represent average ranking for movie $i$:

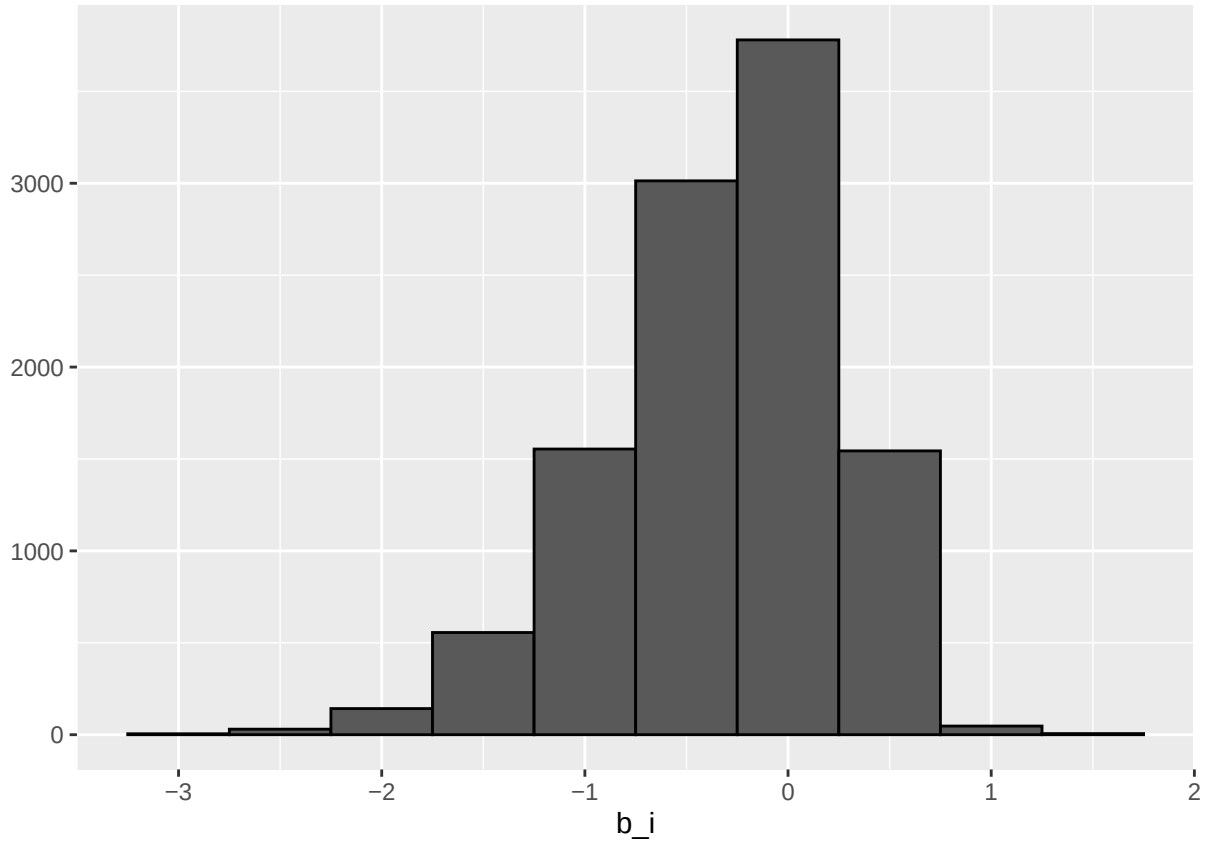
$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

These $b$ notation are referred to as *effects* or *bias* We know that the least square estimate $b_i$ is just the average of $Y_{u,i} - \hat{\mu}$ for each movie $i$. We can compute them as follows:

```
# Model 2: modeling movie effect
# estimate movie bias 'b_i' for all movies
mu <- mean(edx$rating)
movie_avgs <- edx %>%
                group_by(movieId) %>%
                summarize(b_i = mean(rating - mu))
```

We can see that these estimates vary substantially:

```
 # plot these movie 'bias'
qplot(b_i, data = movie_avgs, bins = 10, color = I("black"))
```




Note that $\hat{\mu} = 3.5$ so a $\hat{b_i} = 1.5$ implies a perfect five star rating. Let's see how much our prediction improves once we use $\hat{y_{u,i}} = \hat{\mu} + \hat{b_i}$:

```
# calculate predictions considering movie effect
predicted_ratings <- mu + validation %>%
                      left_join(movie_avgs, by='movieId') %>%
                      pull(b_i)
# calculate rmse after modeling movie effect
model_1_rmse <- RMSE(predicted_ratings, validation$rating)


rmse_results <- bind_rows(rmse_results,
       data_frame(method="Movie Effect Model",
```

```
                    RMSE = model_1_rmse))
rmse_results %>% knitr::kable()
```

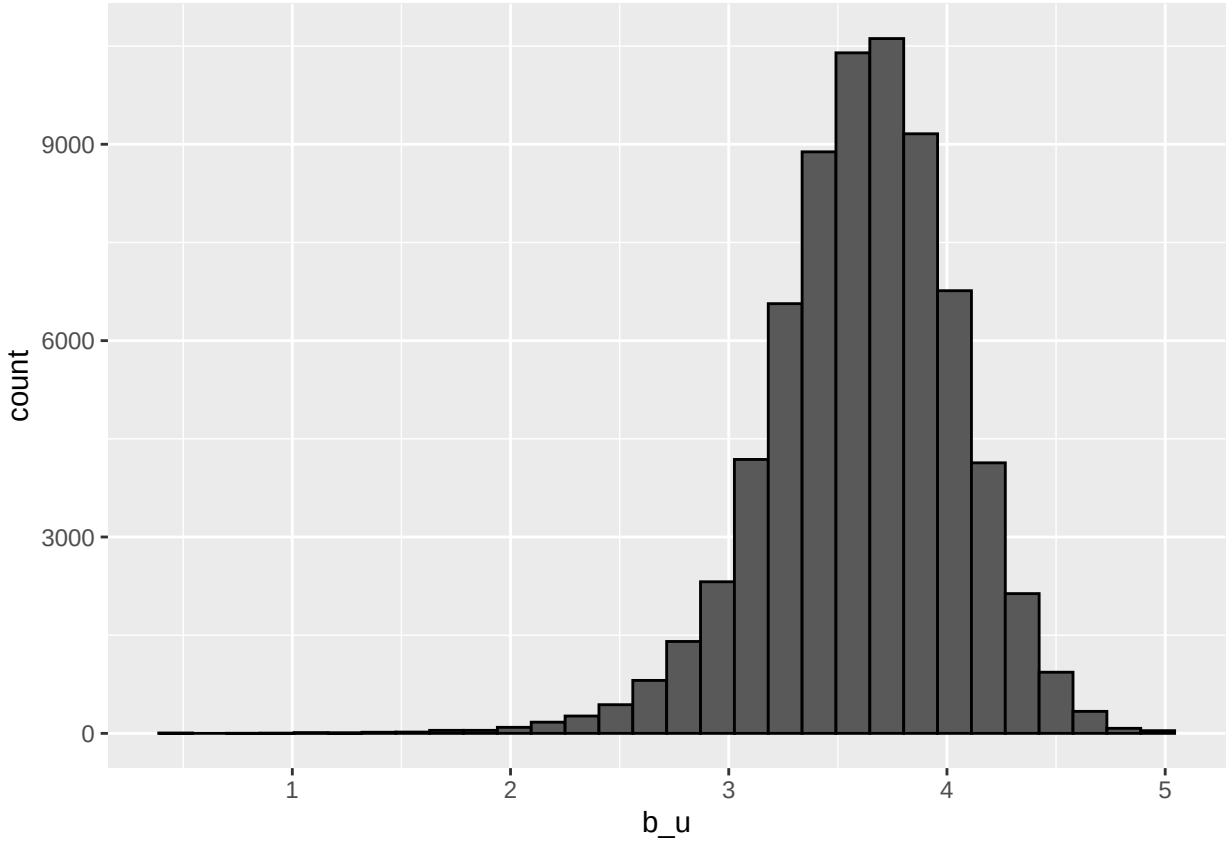
| method | RMSE |
|---|---|
| Single Value Mean | 1.0612018 |
| Movie Effect Model | 0.9439087 |

We already see an improvement. But can we make it better?

### Third model: Modeling User Effects

Let's compute the average rating for user $u$ for those that have rated over 100 movies:

```
# Model 3: modeling user effect in previous model
# plot of avg rating for users that've rated over 100 movies
edx %>% group_by(userId) %>%
            summarize(b_u = mean(rating)) %>%
            filter(n() >= 100) %>%
            ggplot(aes(b_u)) +
            geom_histogram(bins = 30, color = "black")
```




Notice that there is substantial variability across users as well: some users are very cranky and others love every movie. This implies that a further improvement to our model may be:

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

where $b_u$ is a user-specific effect. Now if a cranky user (negative $b_u$) rates a great movie (positive $b_i$), the effects counter each other and we may be able to correctly predict that this user gave this great movie a 3 rather than a 5.

We will compute an approximation by computing $\hat{\mu}$ and $\hat{b_i}$ and estimating $\hat{b_u}$ as the average of $y_{u,i} - \hat{\mu} - \hat{b_i}$:

```
# estimate user bias 'b_u' for all users
user_avgs <- edx %>%
                left_join(movie_avgs, by='movieId') %>%
                group_by(userId) %>%
                summarize(b_u = mean(rating - mu - b_i))
```

We can now construct predictors and see how much the RMSE improves:

```
# calculate predictions considering user effects in previous model
predicted_ratings <- validation %>%
                        left_join(movie_avgs, by='movieId') %>%
                        left_join(user_avgs, by='userId') %>%
                        mutate(pred = mu + b_i + b_u) %>%
                        pull(pred)
# calculate rmse after modeling user specific effect in previous model
model_2_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                data_frame(method="Movie + User Effect Model",
                           RMSE = model_2_rmse))
rmse_results %>% knitr::kable()
```

| method | RMSE |
|---|---|
| Single Value Mean | 1.0612018 |
| Movie Effect Model | 0.9439087 |
| Movie + User Effect Model | 0.8653488 |

## Fourth model: Regularizing Movie and User Effects

### Regularization

The general idea behind regularization is to constrain the total variability of the effect sizes. Why does this help? Consider a case in which we have movie $i = 1$ with 100 user ratings and 4 movies $i = 2, 3, 4, 5$ with just one user rating. We intend to fit the model

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

Suppose we know the average rating is, say, $\mu = 3$. If we use least squares, the estimate for the first movie effect $b_1$ is the average of the 100 user ratings, $1/100 \sum_{i=1}^{100}(Y_{i,1} - \mu)$, which we expect to be a quite precise. However, the estimate for movies 2, 3, 4, and 5 will simply be the observed deviation from the average rating $\hat{b}_i = Y_{u,i} - \hat{\mu}$ which is an estimate based on just one number so it won't be precise at all. Note these estimates make the error $Y_{u,i} - \mu + \hat{b}_i$ equal to 0 for $i = 2, 3, 4, 5$, but this is a case of over-training. In fact, ignoring the one user and guessing that movies 2,3,4, and 5 are just average movies ($b_i = 0$) might provide a better

prediction. The general idea of penalized regression is to control the total variability of the movie effects: $\sum_{i=1}^{5} b_i^2$. Specifically, instead of minimizing the least squares equation, we minimize an equation that adds a penalty:

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

The first term is just least squares and the second is a penalty that gets larger when many $b_i$ are large. Using calculus we can actually show that the values of $b_i$ that minimize this equation are:

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

where $n_i$ is the number of ratings made for movie $i$. This approach will have our desired effect: when our sample size $n_i$ is very large, a case which will give us a stable estimate, then the penalty $\lambda$ 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 $\lambda$, the more we shrink.

### Regularizing Movie and User Effects

We can use regularization for the estimate user effects as well. We are minimizing:

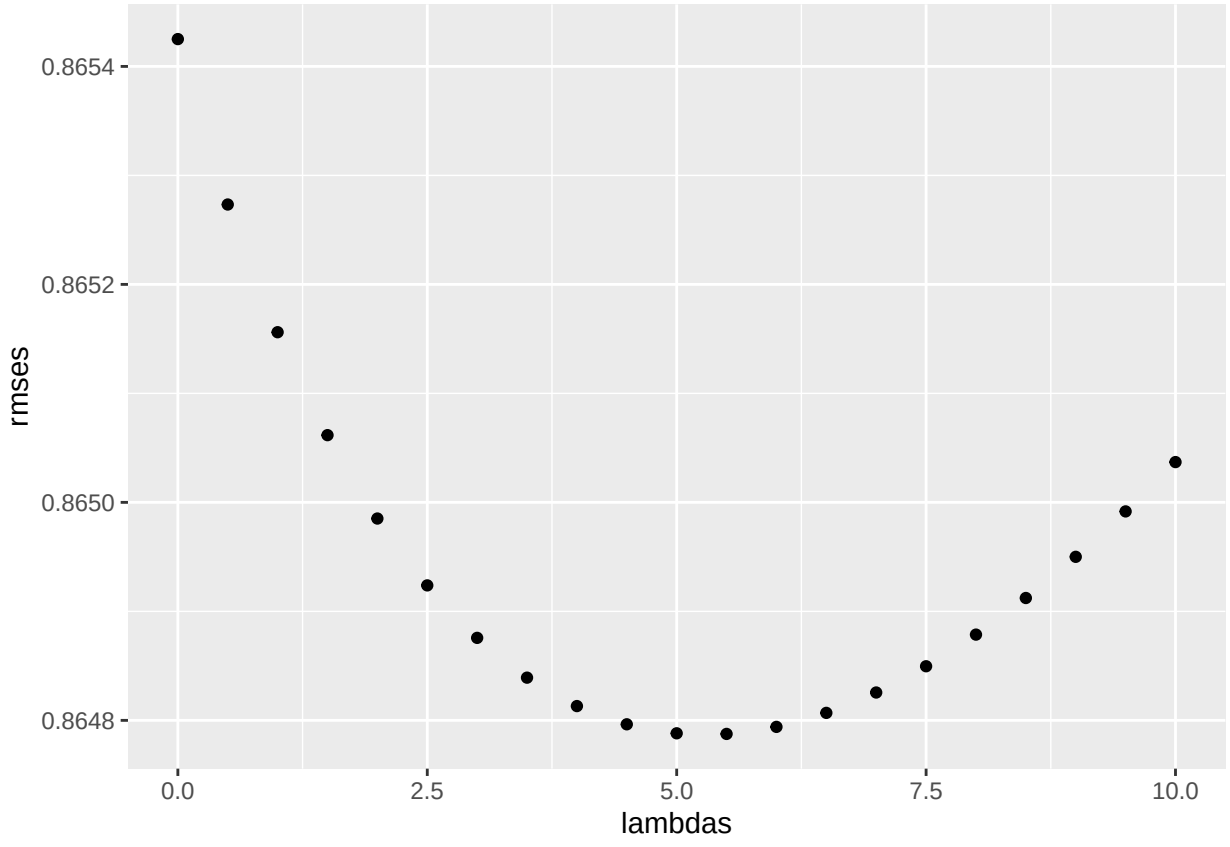
$$\frac{1}{N}\sum_{u,i}\left(y_{u,i}-\mu-b_i-b_u\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right)$$

Note that $\lambda$ is a tuning parameter. We can use cross-validation to choose it.

For cross-validation we use additional partition from the *edx* test set.

```
 # Model 4: regularizing movie + user effect model from previous models
 # choosing the penalty term lambda

 # Create additional Partition from the edx test set into edx_test and edx_val to obtain lambda
 test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
 edx_test <- edx[-test_index,]
 edx_temp <- edx[test_index,]

 # Make sure userId and movieId in validation set are also in edx_test set
edx_val <- edx_temp %>%
  semi_join(edx_test, by = "movieId") %>%
  semi_join(edx_test, by = "userId")


# Add rows removed from validation set back into edx_test set
removed <- anti_join(edx_temp, edx_val)
edx_test <- rbind(edx_test, removed)

 #

 lambdas <- seq(0, 10, 0.5)
```

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

  mu <- mean(edx_test$rating)

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

  b_u <- edx_test %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  predicted_ratings <-
      edx_val %>%
      left_join(b_i, by = "movieId") %>%
      left_join(b_u, by = "userId") %>%
      mutate(pred = mu + b_i + b_u) %>%
      pull(pred)

  return(RMSE(predicted_ratings, edx_val$rating))
})


qplot(lambdas, rmses)
```



### Optimal Lambda and Regularized Model RMSE

Based on cross-validation from additional partition, the optimal $\lambda$ is:

```
 # lambda that minimizes rmse
 lambda <- lambdas[which.min(rmses)]
 lambda
```

```
## [1] 5.5
```

So, $\lambda$ is the penalty term that minimizes the RMSE based on the validation done on the sub-partition datasets We use this lambda to run the model against our original *edx* (train) and *validation* set.

```
# Using the optimal lambda, run the model with the original edx train and validation set
   mu <- mean(edx$rating)

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

   b_u <- edx %>%
     left_join(b_i, by="movieId") %>%
     group_by(userId) %>%
     summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

   predicted_ratings <-
       validation %>%
       left_join(b_i, by = "movieId") %>%
       left_join(b_u, by = "userId") %>%
       mutate(pred = mu + b_i + b_u) %>%
       pull(pred)

   RMSE_Regularized <- RMSE(predicted_ratings, validation$rating)


 # calculate rmse after regularizing movie + user effect from previous models
 rmse_results <- bind_rows(rmse_results,
data_frame(method="Regularized Movie + User effect model",
           RMSE = RMSE_Regularized))
```

# Results

We can inspect the RMSEs for the various model trained from naive approach using the mean, to regularized movie and user effects model using optimal value of tuning parameter $\lambda$.

The RMSE result table for various models are as follows:

| method | RMSE |
|---|---|
| Single Value Mean | 1.0612018 |
| Movie Effect Model | 0.9439087 |
| Movie + User Effect Model | 0.8653488 |
| Regularized Movie + User effect model | 0.8648178 |

As we see from the table the RMSEs of each model is an improvement from the previous models, with the last model having an RMSE $< 0.86490$. This is obviously a significant improvement from the first naive model.

# Conclusion

The RMSEs improved as we include additional features to the machine learning algorithm. *Regularized Movie and User Effect Model* provided the lowest RMSE value $< 0.86490$. The simplest single value mean model that predicts the same rating for all movies regardless of user gave an RMSE $> 1$. If RMSE is larger than 1, it means our typical error is larger than one star, which is not good. By taking into consideration the *movie effect* the RMSE went down $< 1$ which was a great improvement. RMSE further went down after modeling the *user effect*. However, the lowest RMSE $< 0.86490$ was achieved by regularizing the movie and user effect.

## Future Impovement

Regularizing the *year* and *genres* effects can further improve the RMSE, but regularizing the *genres* is computationally very expensive since it requires separating multiple genres for many movies into multiple observations for a given movie $i$ having single genre in *genres* column.

### References


- https://bits.blogs.nytimes.com/2009/09/21/netflix-awards-1-million-prize-and-starts-a-new-contest
- http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary
- https://www.netflixprize.com/assets/GrandPrize2009_BPC_BellKor.pdf