Capstone Project: MovieLens

Vasu Goel

10/2/2019

Introduction

This Capstone Project is part of the edx course HarvardX:PH125.9x Data Science: Capstone and is based on the challenge that Netflix offered to the data science community.The challenge was to improve Netflix's movie recommendation system by 10%. The Netflix movie rating scheme uses a rating range from 0 to 5. In this project the objective is to train a machine learning algorithm using inputs from one data set to predict movie ratings in another data set. The data set from which the two sets mentioned previously will be created can be found using the following link:

http://files.grouplens.org/datasets/movielens/ml-10m.zip

Recommendation systems use ratings that users have given items to make specific recommendations. Companies that sell many products to many customers and permit these customers to rate their products, like Amazon, are able to collect massive datasets that can be used to predict what rating a particular user will give a specific item. Items for which a high rating is predicted for a given user are then recommended to that user. Netflix uses a recommendation system to predict how many stars a user will give a specific movie. Based on the predictions, the movie with the highest predicted rating for a user is then recommended to the user.

Although predicting ratings seems easy, there are many different biases in the movie ratings. Many users prefer different genres than others, some only rate the movies they liked and others only the movies they disliked. In addition, many movies get a great review most of the time. These biases can be taken into consideration using various machine learning models.

The Netflix challenge used the typical error loss: they decided on a winner based on the residual mean squared error (RMSE) on a test set. To provide satisfying results many different models were trained for this project and the best model was chosen based on the RMSE on the validation set.

Getting Data

First step in any data analysis project is to get the data. The Netflix data is not publicly available, but the GroupLens research lab generated their own database with over 20 million ratings for over 27,000 movies by more than 138,000 users. In this analysis, a smaller subset of about 10 million ratings was used.

We can download the 10M version of MovieLens dataset used in the analysis using this code:

```r
# 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")

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip
```

```
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")
```

Create Train and Validation Sets

Use the following code to generate the datasets. The algorithm was developed using the edx set. For a final test of the algorithm we predict movie ratings in the validation set as if they were unknown.

```
# 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)
```

MovieLens Data

MovieLens dataset initially had 10 million ratings (rows) with 6 variables (columns) which were partitioned into *edx* and *validation* sets containing ~9M and ~1M ratings respectively.

We can see the sets are in tidy format:

```
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
```

```
validation %>% as_tibble()
```

```
## # A tibble: 999,999 x 6
##    userId movieId rating  timestamp title                 genres
##     <int>   <dbl>  <dbl>      <int> <chr>                 <chr>
##  1      1     231    5    838983392 Dumb & Dumber (1994)  Comedy
##  2      1     480    5    838983653 Jurassic Park (1993)  Action|Adventure~
##  3      1     586    5    838984068 Home Alone (1990)     Children|Comedy
##  4      2     151    3    868246450 Rob Roy (1995)        Action|Drama|Rom~
##  5      2     858    2    868245645 Godfather, The (1972) Crime|Drama
##  6      2    1544    3    868245920 Lost World: Jurassic~ Action|Adventure~
##  7      3     590    3.5 1136075494 Dances with Wolves (~ Adventure|Drama|~
##  8      3    4995    4.5 1133571200 Beautiful Mind, A (2~ Drama|Mystery|Ro~
##  9      4      34    5    844416936 Babe (1995)           Children|Comedy|~
## 10      4     432    3    844417070 City Slickers II: Th~ Adventure|Comedy~
## # ... with 999,989 more rows
```

Data Exploration

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

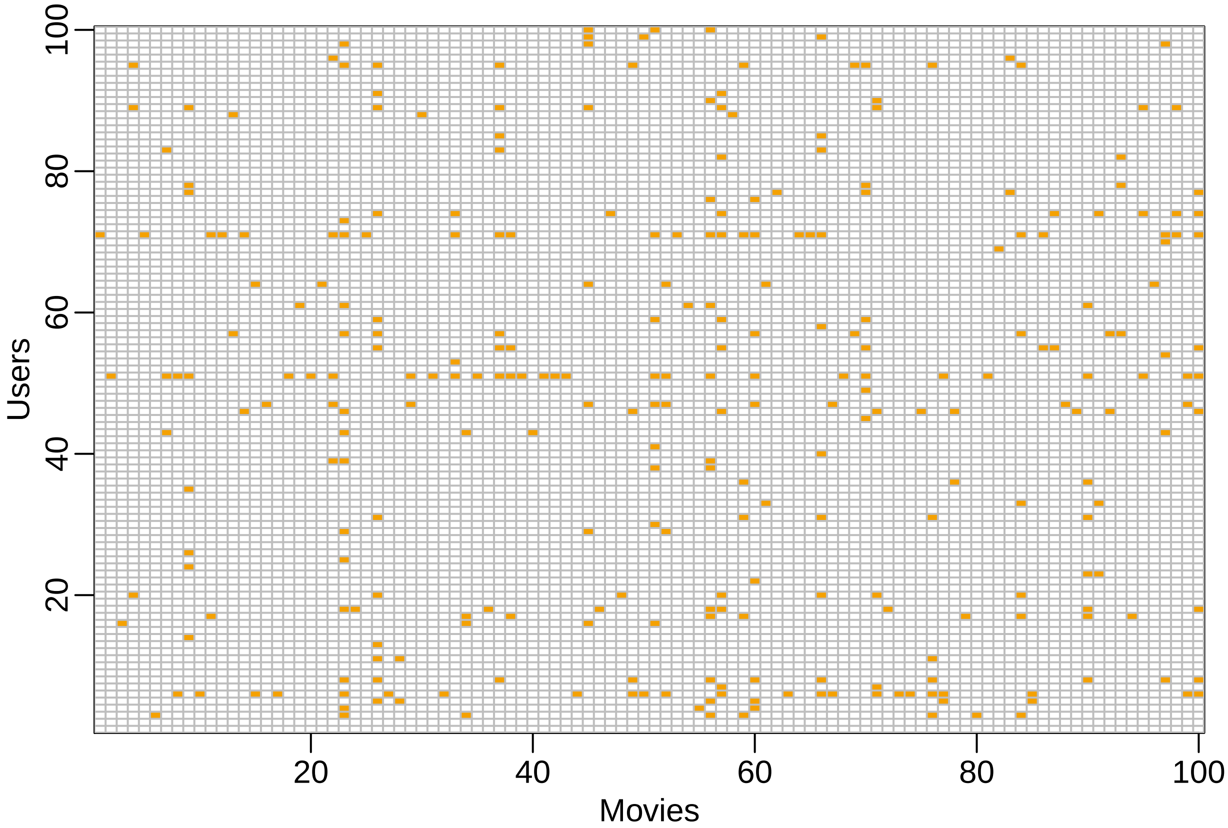
```
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   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 user u in rows and movie i in columns.

You can think of the task of a recommendation system as filling in those missing 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.

```
users <- sample(unique(edx$userId), 100)
rafalib::mypar()
edx %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  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")
```



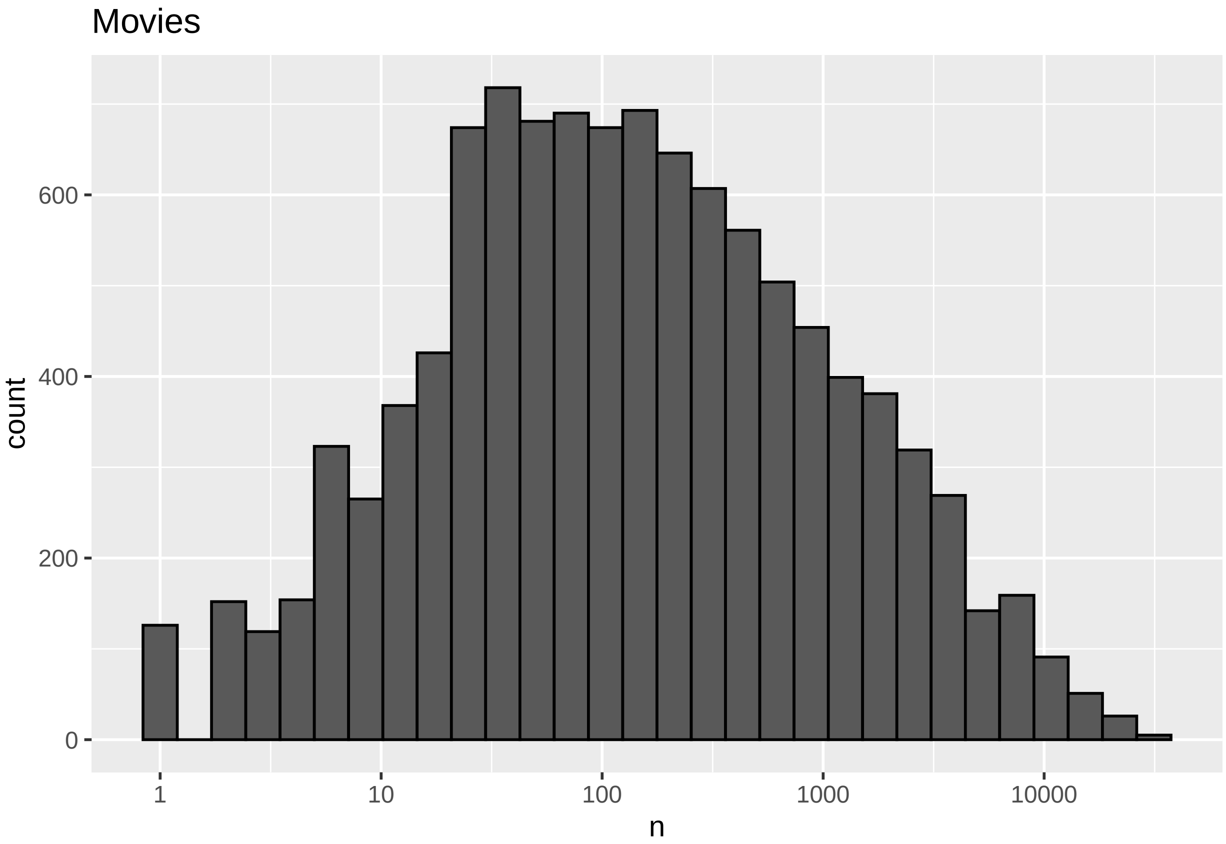
Note that if we are predicting the rating for movie i by user u , in principle, all other ratings related to movie i and by user u may used as predictors, but different users rate different movies and a different number of movies. Furthermore, we may be able to use information from other movies that 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.

Distributions

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

The first thing we notice is that some movies get rated more than others. Here is the distribution:

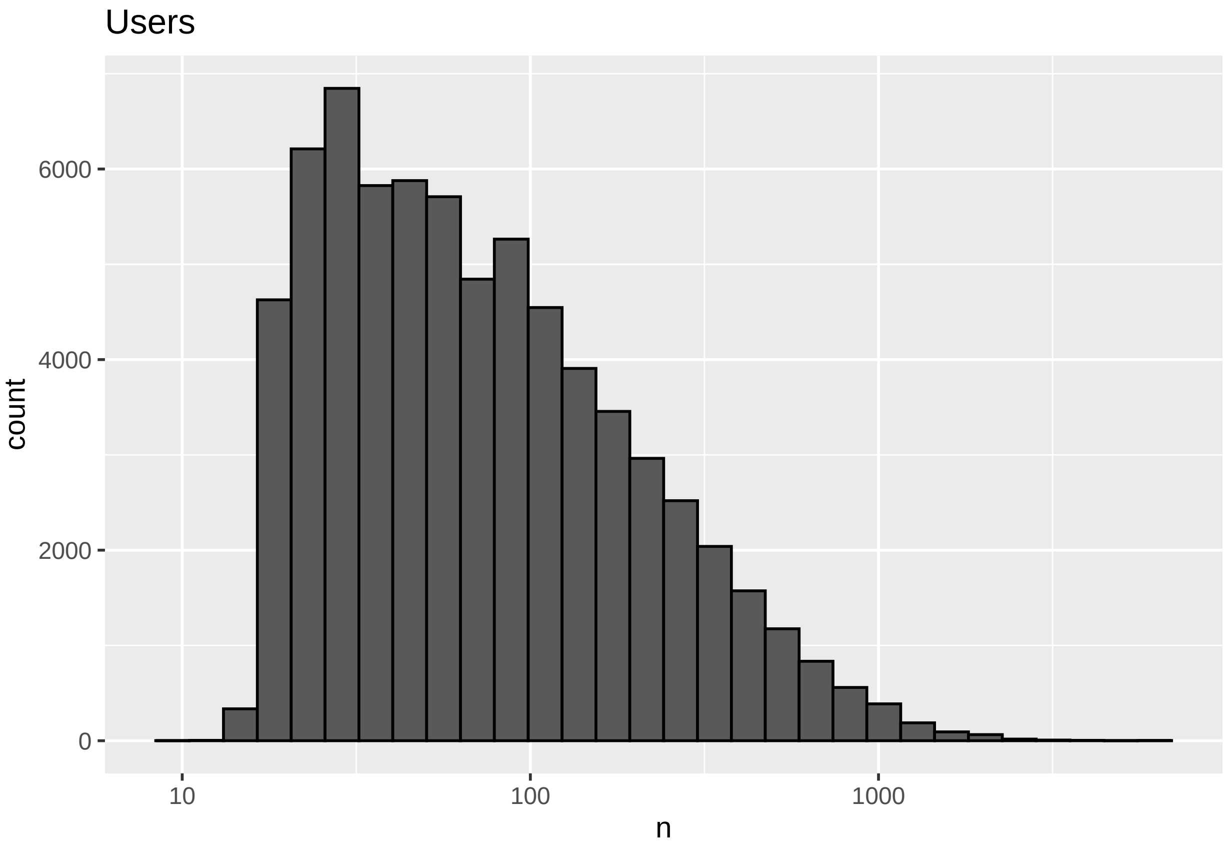
```
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies")
```



This should not surprise us given that there are blockbuster movies watched by millions and artsy, independent movies watched by just a few.

Our second observation is that some users are more active than others at rating movies:

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



Data Analysis

Loss Function

We use residual mean squared error (RMSE) as the metric to determine how good our algorithm performed. If we define $y_{u,i}$ as the rating for movie i by user u and denote out prediction with $\hat{y}_{u,i}$. The RMSE is then 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 interpretted 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:

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

First model: Naive approach

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 μ the "true" rating for all movies. We know that the estimate that minimizes the RMSE is the least squares estimate of μ and, in this case, is the average of all ratings:

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

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

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

```r
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:

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

method	RMSE
Naive approach	1.061202

Second model: Modelling 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:

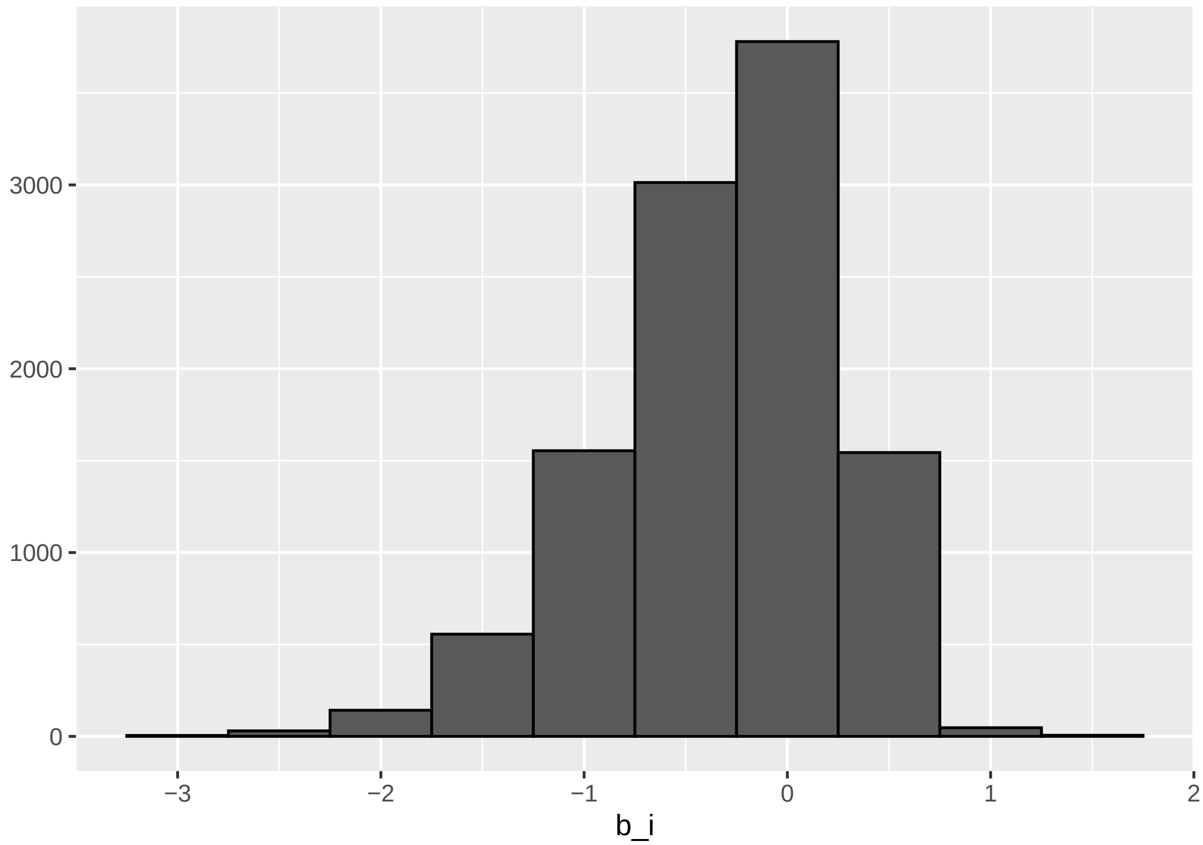
$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

These bs are referred to as *effects* or *bias*, thus the b notation. 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:

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

We can see that these estimates vary substantially:

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



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 modelling 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()
```

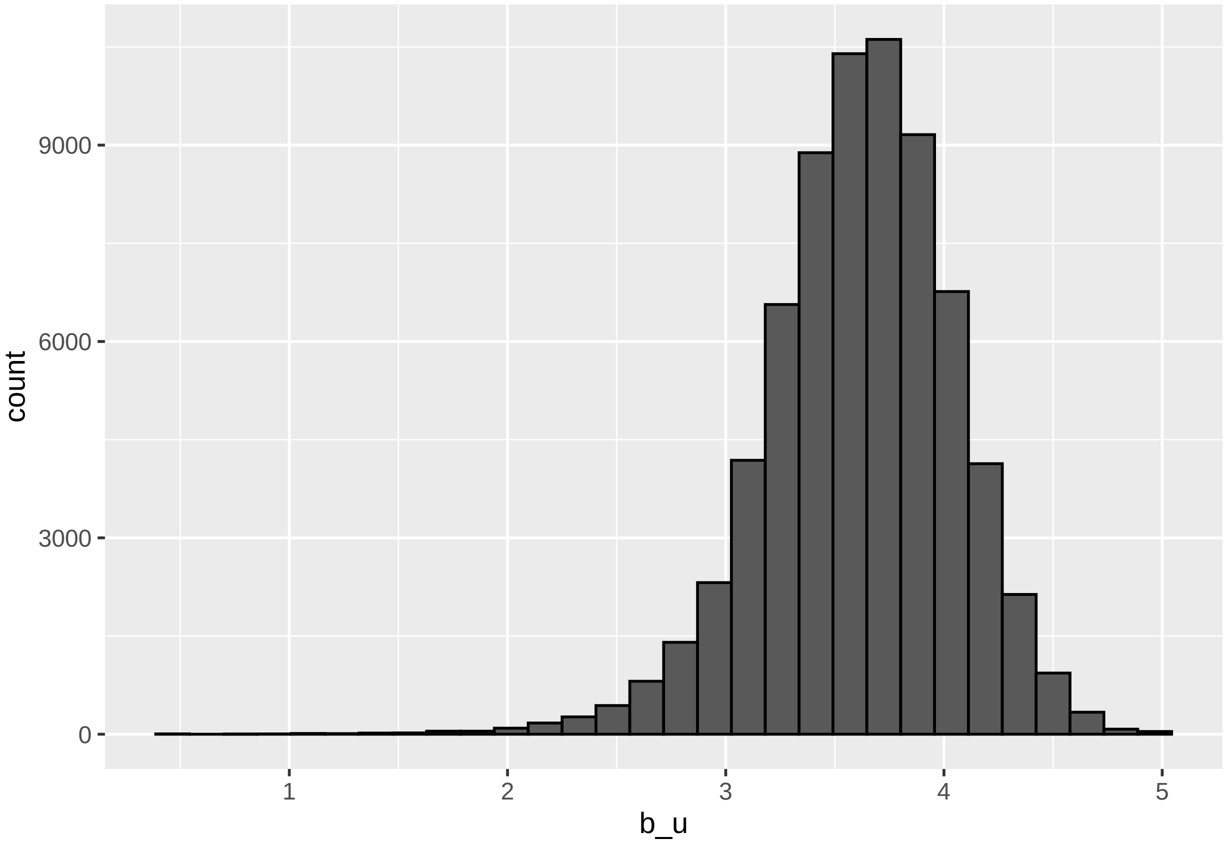
method	RMSE
Naive approach	1.0612018
Movie effect model	0.9439087

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

Third model: Modelling user effects

Let's compute the average rating for user u for those that have 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 $y_{u,i} - \hat{\mu} - \hat{b_i}$:

```
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 modelling 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 effects model",
                                     RMSE = model_2_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Naive approach	1.0612018
Movie effect model	0.9439087
Movie + User effects 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:

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

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 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 we shrink.

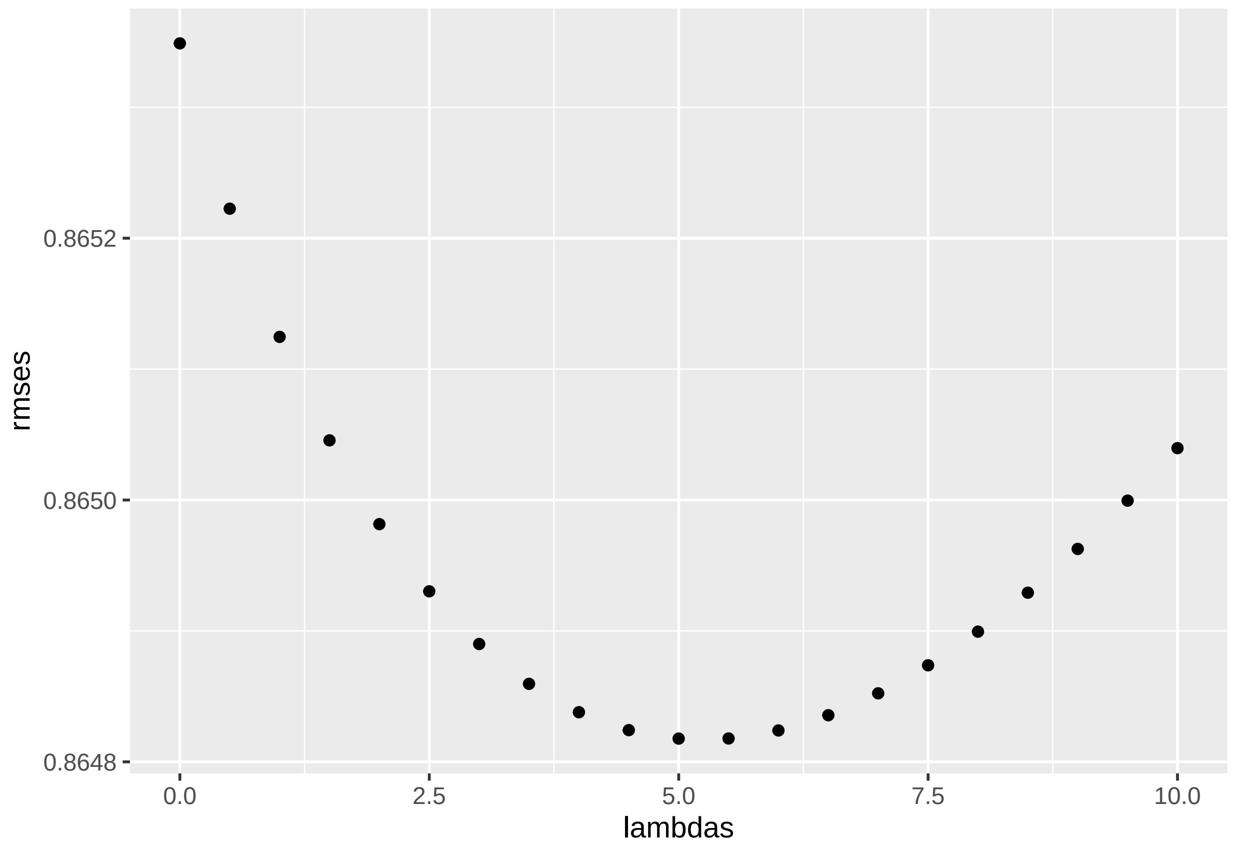
Regularizing movie and user effects

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

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

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

```r
# choosing the penalty term lambda
lambdas <- seq(0, 10, 0.5)

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

  mu <- mean(edx$rating)

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

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

  predicted_ratings <-
      validation %>%
      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, validation$rating))
})

qplot(lambdas, rmses)
```



For the full model, the optimal λ is:

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

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

So, $\lambda = 5$ is the penalty term that minimizes the RMSE.

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

method	RMSE
Naive approach	1.0612018
Movie effect model	0.9439087
Movie + User effects model	0.8653488
Regularized Movie + User effect model	0.8648177

Results

We can inspect the RMSEs for the various model trained from naive approach (of predicting the mean rating regardless of the user) to regularized movie and user effects (by finding the optimal value of turing parameter λ).

The resultant RMSEs for various models are as follows:

method	RMSE
Naive approach	1.0612018
Movie effect model	0.9439087
Movie + User effects model	0.8653488
Regularized Movie + User effect model	0.8648177

As we see from the results each model is an improvement from the previous one with the last one having an RMSE lower than 0.8649. This is obviously a significant improvement from the first naive model. With the help of the last model our machine learning algorithm was able to predict movie ratings for the movies in validation set with an RMSE of about 0.8648201 i.e, we are within 0.8648201 of the "true" ratings.

Conclusion

The simplest model which predicts the same rating (mean rating) for all movies regardless of user gave an RMSE above 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 to 0.9439087 which was a great improvement. RMSE further went down to 0.8653488 after modelling the *user effect* in the previous model. However, the lowest RMSE i.e, 0.8648201 was achieved by regularizing the movie and user effect, which penalized larger estimates of ratings from relatively small sample size of users.

Subsequently, regularizing the *year* and *genres* effects can further improve the residual mean squared error but this process is computationally expensive.