MovieLens

Gladys Teeson

6/5/2020

Contents

Introduction **2**

MovieLens Dataset **2**

Load Libraries 2

Import Dataset 2

Generate Train and Validation Sets 3

Tidy Format of Data 3

Data Wrangling 3

Exploratory Data Analysis 4

Machine Learning Data Models **7**

RMSE - The Loss Function 7

Model One: Simple Average Rating 8

Model Two: Movie Effects 8

Model Three: Movie and User Effects 10

Model Four: Regularization 11

Regularization of Movie and User Effects 13

Regularized RMSE using Lambda 14

Results **15**

Conclusion **15**

Future Impovement 16

References 16

Introduction

Recommendation systems use ratings that *users* have given *items* to make specific recommendations. Items for which a high rating is predicted for a given user are then recommended to that user. For example, Netflix uses a recommendation system to predict how many stars a user will give a specific movie. One star suggests it is not a good movie, whereas five stars suggests it is an excellent movie.

For this project, we will be creating a movie recommendation system using the MovieLens dataset. We will use the 10M version of the MovieLens dataset to make the computation a little easier. The objective is to model an algorithm that predicts movie ratings with the least residual mean squared error (RMSE). i.e below 0.86490.

MovieLens Dataset

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:

Load Libraries

```
#################################
# 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(lubridate)) install.packages("lubridate", 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
```

Import Dataset

```
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")
```

Generate Train and Validation Sets

We will now generate the *Train* and *Validation* sets in the ratio 90:10. While the *Train* set is used for training and testing the various models, the *Validation* set is only used for determining the RMSE of the models.

```r
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)
```

Tidy Format of Data

The MovieLens dataset is in a tidy format with row representing a rating given by one user to one movie and the column representing the features for each rating. We see the edx *train* table is in tidy format with millions of rows and 6 columns:

```r
# tidy format of 'edx' data
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 Wrangling

In the edx dataset we can see that `timestamp` column is not in a readable format. We will convert the timestamp into date and years. We will also extract the years fom the `title` column to a separate column.

```r
# Convert timestamp to date format using lubridate function, add columns for rating_year and release_ye
edx_mut <- edx %>% mutate(timestamp = as_datetime(timestamp), rating_year = year(as_datetime(timestamp)
head(edx_mut)
```

```
##    userId movieId rating           timestamp                         title
## 1:      1     122      5 1996-08-02 11:24:06              Boomerang (1992)
## 2:      1     185      5 1996-08-02 10:58:45               Net, The (1995)
## 3:      1     292      5 1996-08-02 10:57:01               Outbreak (1995)
## 4:      1     316      5 1996-08-02 10:56:32               Stargate (1994)
## 5:      1     329      5 1996-08-02 10:56:32 Star Trek: Generations (1994)
## 6:      1     355      5 1996-08-02 11:14:34       Flintstones, The (1994)
##                           genres rating_year release_year
## 1:                Comedy|Romance        1996         1992
## 2:         Action|Crime|Thriller        1996         1995
## 3:  Action|Drama|Sci-Fi|Thriller        1996         1995
## 4:       Action|Adventure|Sci-Fi        1996         1994
## 5: Action|Adventure|Drama|Sci-Fi        1996         1994
## 6:       Children|Comedy|Fantasy        1996         1994
```

Now we have a new *train* dataset *edx_mut* with 8 columns:

```r
# structure of our new edx dataset
str(edx_mut)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  8 variables:
##  $ userId      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId     : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating      : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp   : POSIXct, format: "1996-08-02 11:24:06" "1996-08-02 10:58:45" ...
##  $ title       : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres      : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Actio
##  $ rating_year : num  1996 1996 1996 1996 1996 ...
##  $ release_year: num  1992 1995 1995 1994 1994 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Exploratory Data Analysis

Each row in the dataset 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:

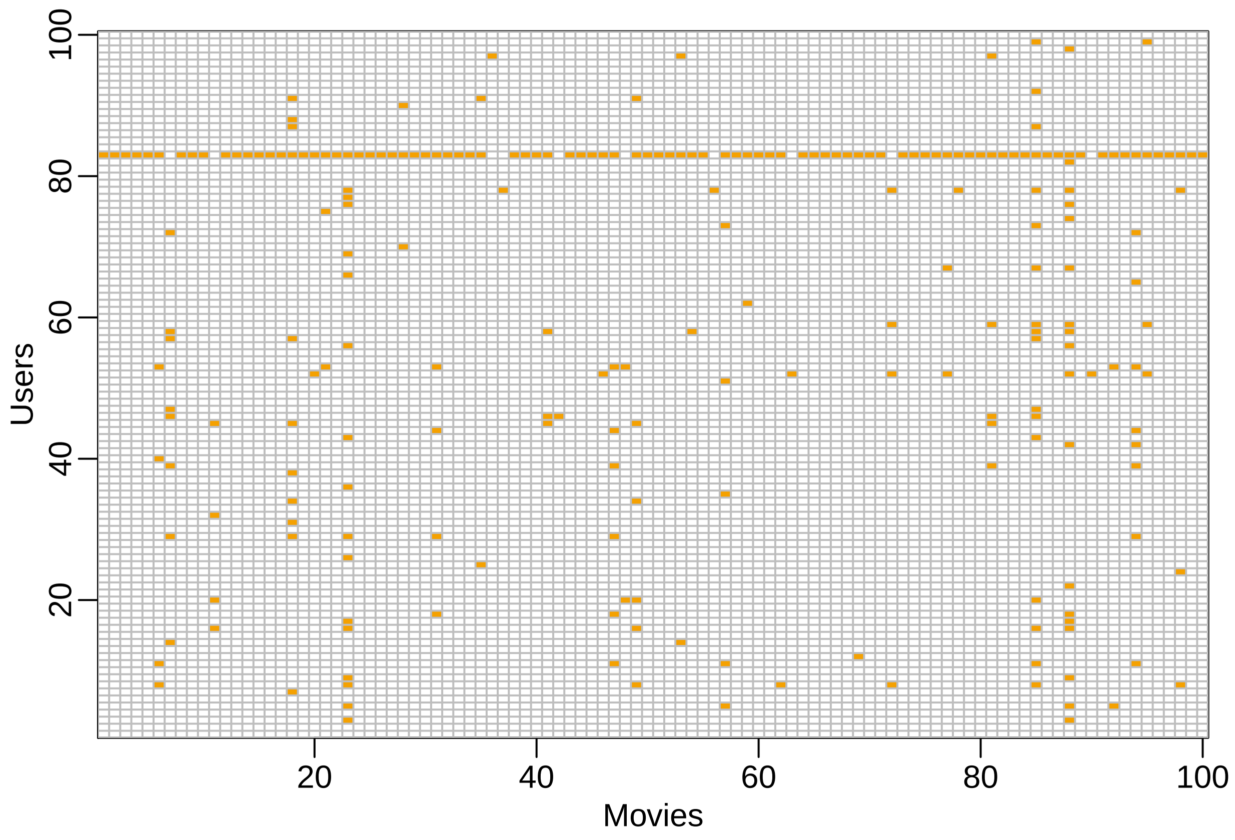
```r
# number of unique users and movies
edx_mut %>% summarize(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

```
##   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.

```r
# 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")
```

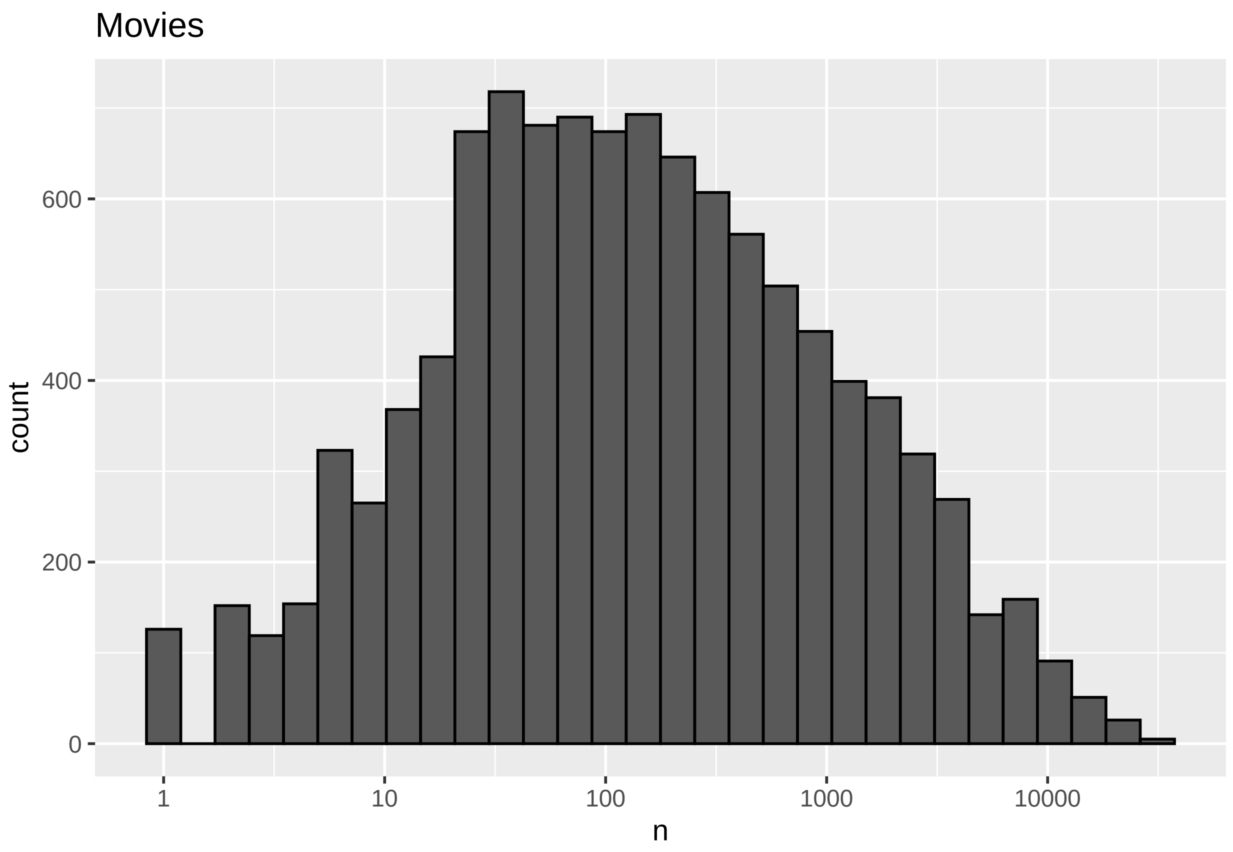


Note that if we are predicting the rating for movie i by user u , in principle, all other ratings related to movie i and by user u may used as predictors, but different users rate different movies and a different number of movies. Furthermore, we may be able to use information from other movies that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

Now, we will analyse the general properties of the data.

Firstly, 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:

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



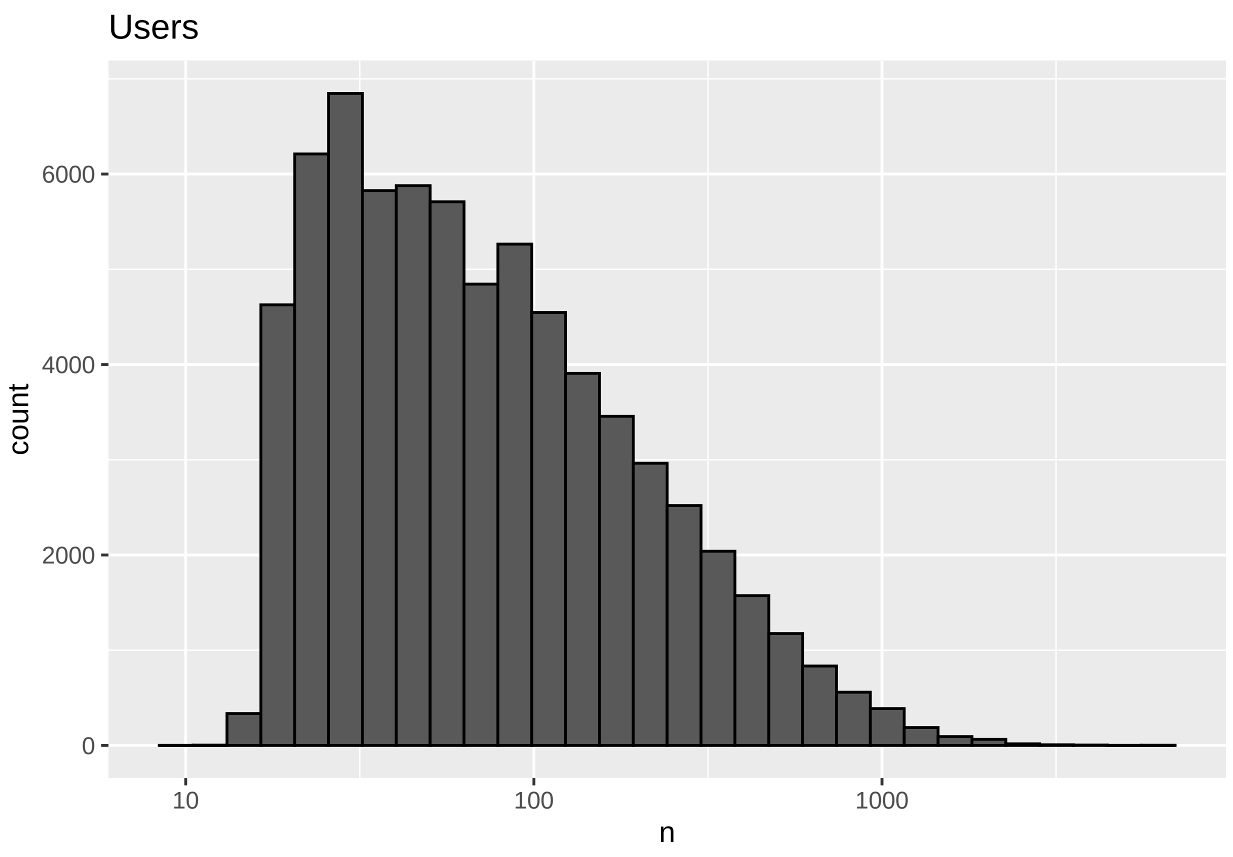
Let's look at the top 10 movies with the most ratings per year since its release. This is to confirm that blockbusters are rated more often.

```
# top 10 movies with the most ratings per year since its release
years_rated_max <- max(edx_mut$rating_year) + 1
edx_mut %>% group_by(movieId) %>%
  summarize(n = n(), title = title[1], years = years_rated_max - first(release_year), avg_rating = mean
  mutate(rate = n/years) %>%
  top_n(10, rate) %>%
  arrange(desc(rate))
```

```
## # A tibble: 10 x 6
##    movieId     n title                                 years avg_rating  rate
##      <dbl> <int> <chr>                                 <dbl>      <dbl> <dbl>
##  1     296 31362 Pulp Fiction (1994)                      16       4.15 1960.
##  2     356 31079 Forrest Gump (1994)                      16       4.01 1942.
##  3    2571 20908 Matrix, The (1999)                       11       4.20 1901.
##  4    2858 19950 American Beauty (1999)                   11       4.19 1814.
##  5     318 28015 Shawshank Redemption, The (1994)         16       4.46 1751.
##  6     110 26212 Braveheart (1995)                        15       4.08 1747.
##  7     480 29360 Jurassic Park (1993)                     17       3.66 1727.
## 8     780 23449 Independence Day (a.k.a. ID4) (1996)        14       3.38 1675.
## 9    5952 12969 Lord of the Rings: The Two Towers, The ~     8       4.12 1621.
## 10    150 24284 Apollo 13 (1995)                            15       3.89 1619.
```

Secondly, we notice that some users are more active than others at rating movies. Below is the distribution:

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



Machine Learning Data Models

RMSE - The Loss Function

The typical error loss *residual mean squared error* (RMSE) on the test set is used to determine the performance of our models.

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:

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

Model One: Simple Average Rating

Let's start by building the simplest possible recommendation system: we predict the same rating for all movies regardless of user.

```r
# First Model: same rating for all movies using average rating of all movies

# calculate average rating of all movies
mu_hat <- mean(edx_mut$rating)
mu_hat
```

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

The average rating for all movies across all users is 3.51.

```r
# calculate rmse
naive_rmse <- RMSE(validation$rating, mu_hat)
naive_rmse
```

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

The rmse is larger than one, which means our typical error is larger than one star, which is not good.

We will now create a results table to add this rmse. We will continue to add the rmse's of different models to this same results table.

```r
# add rmse results in a table
rmse_results <- data.frame(method = "Simple Average Rating Model", RMSE = naive_rmse)
rmse_results %>% knitr::kable()
```

method	RMSE
Simple Average Rating Model	1.061202

Model Two: Movie Effects

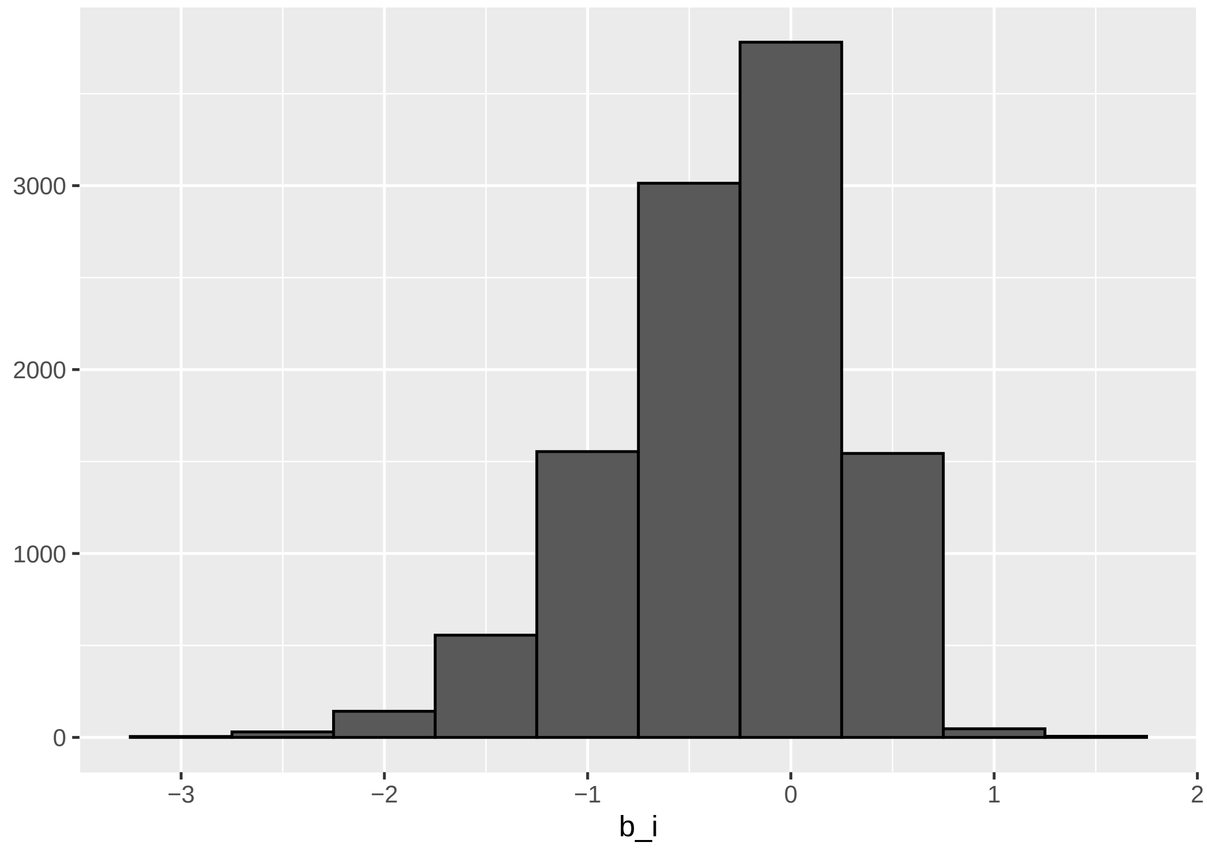
We already examined that some movies are just generally rated higher than others(i.e top 10 highly rated movies). This is the movie effect or bias. We can get this bias by finding the average of the difference between the actual rating and overall average rating (mu) for each movie. We can compute them as follows:

```r
# Second Model: movie effect on average rating

# calculate movie bias 'b_i' for all movies
```

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

We can see that these estimates vary substantially:

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



We can now calculate the predictions and rmse of this model:

```r
 # calculate predictions using movie effect
predicted_ratings <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
# calculate rmse
model_1_rmse <- RMSE(predicted_ratings, validation$rating)
# add rmse results in to the table
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effects Model",
                                     RMSE = model_1_rmse))
rmse_results %>% knitr::kable()
```

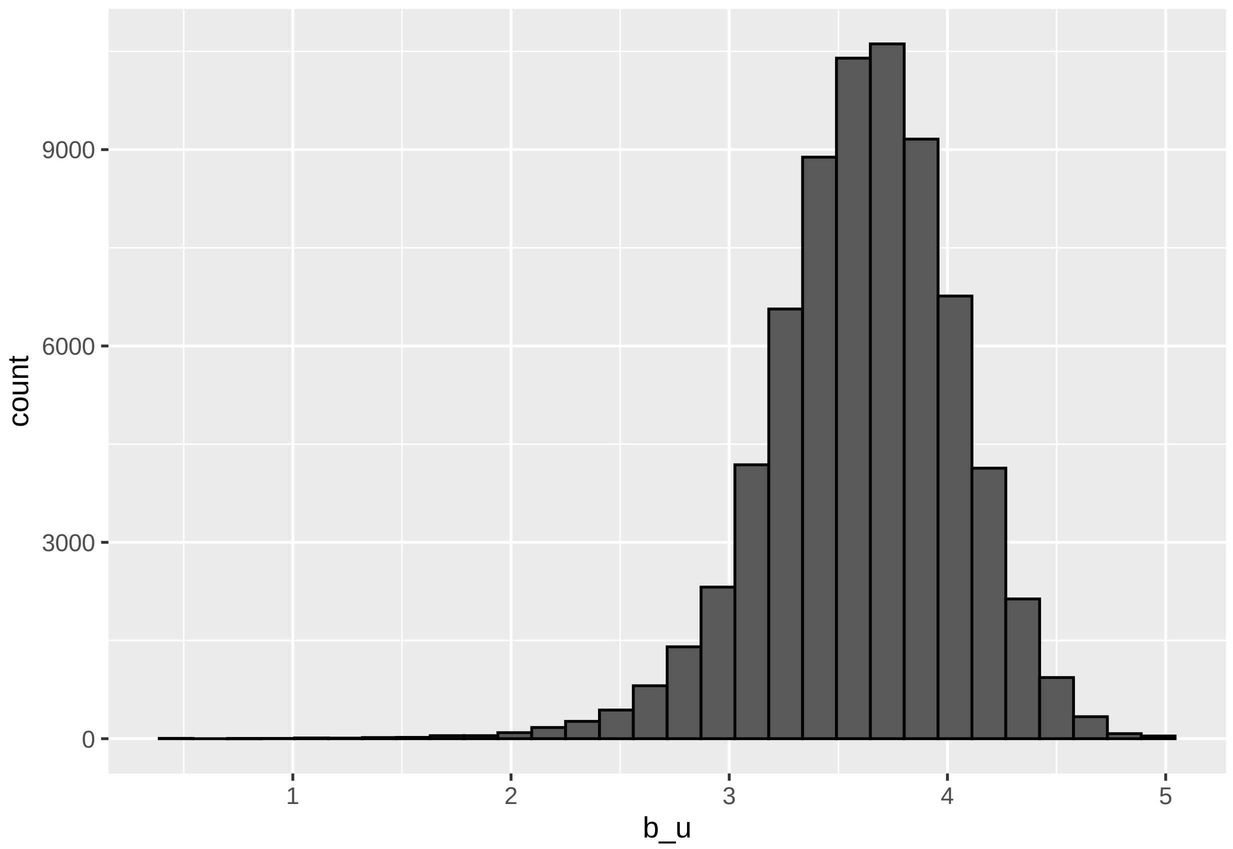
method	RMSE
Simple Average Rating Model	1.0612018
Movie Effects Model	0.9439087

There is an improvement with the movie effect and the rmse is below one. But we can do better.

Model Three: Movie and User Effects

We will now compute the average rating for user u for those that have rated over 100 movies:

```r
# Third Model: combining user effect and movie effect

# plot of avg rating for users that've rated over 100 movies
edx_mut %>% 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. The average rating for each user varies substantially. We can find this user effect as follows:

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

We can now calculate the predictions and rmse of this model:

```r
# calculate predictions using user effects along with movie effects
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
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
Simple Average Rating Model	1.0612018
Movie Effects Model	0.9439087
Movie + User Effects Model	0.8653488

The addition of the user bias again improved our model substantially.

Model Four: Regularization

We will explore the data again to see if we can improve our RMSE. It is possible that some obscure movies with extreme high and low ratings, but with ratings from very few users are impacting our user and movie effect models.

Let's look at the top 10 best movies according to our estimates.

```r
# top 10 and bottom 10 movies according to our estimates
movie_titles <- edx_mut %>%
  select(movieId, title) %>%
  distinct()
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  slice(1:10)  %>%
  pull(title)
```

```
##  [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)"
```

Those are pretty obscure movies. Here are the 10 worst movies:

```
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  slice(1:10)  %>%
  pull(title)
```

```
##  [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)"
```

Here is how often the top movies are rated:

```
# examine how often the top movies are rated
edx_mut %>% count(movieId) %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  slice(1:10) %>%
  pull(n)
```

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

Here is how often the worst movies are rated:

```
# examine how often the worst movies are rated
edx_mut %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  slice(1:10) %>%
  pull(n)
```

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

The supposed "best" and "worst" movies were rated by very few users, in most cases just 1. These movies were mostly obscure ones. This is because with just a few users, we have more uncertainty. Therefore, larger estimates of b_i, negative or positive, are more likely.

These are noisy estimates that we should not trust, especially when it comes to prediction. Large errors can increase our RMSE, so we would rather be conservative when unsure.

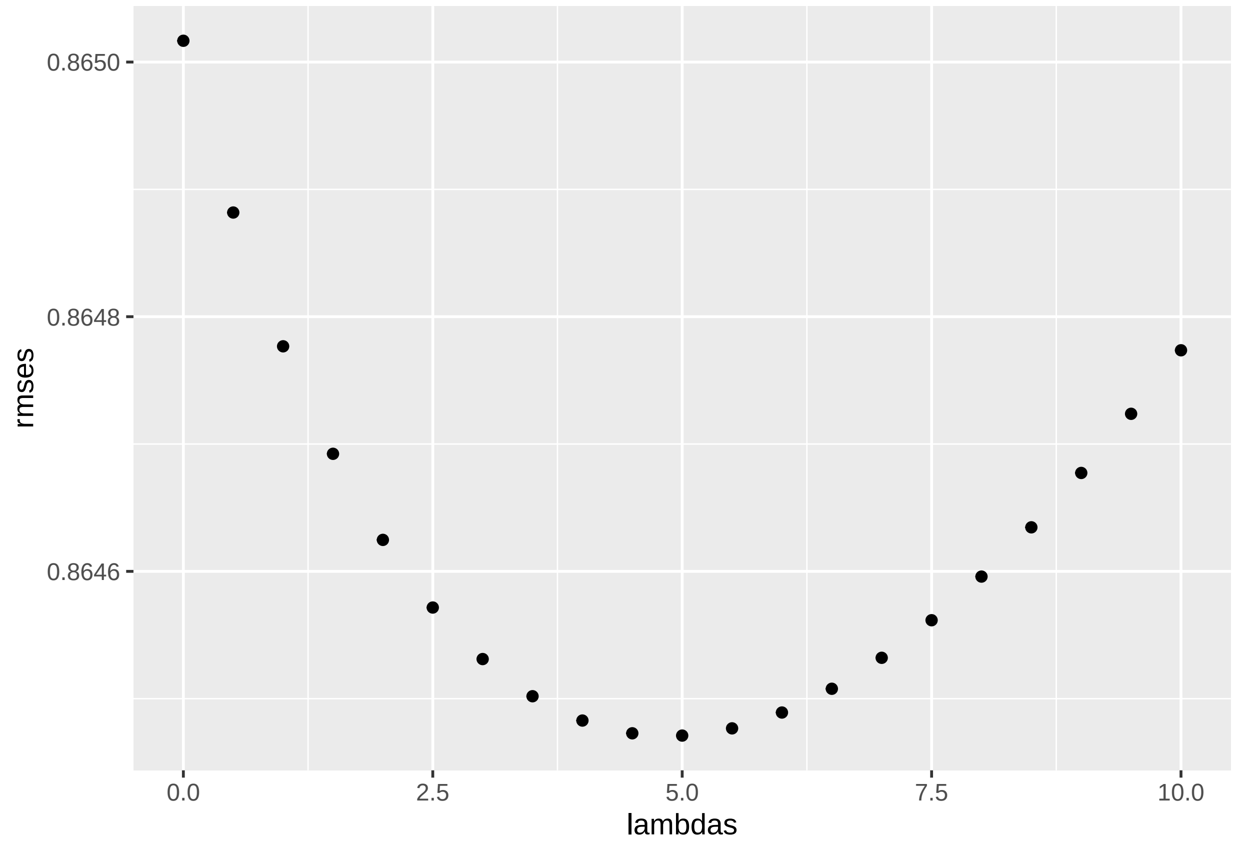
Regularization permits us to penalize large estimates that are formed using small sample sizes.

Regularization of Movie and User Effects

The penalty term lambda is then added to our model. Note that lambda is a tuning parameter. We can use cross-validation to determine the optimal lambda.

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

```
# Fourth Model: regularizing movie + user effect
# choosing the penalty term lambda

# Create additional Partition from the edx_mut test set into edx_mut_test and edx_mut_val to obtain lam
test_index <- createDataPartition(y = edx_mut$rating, times = 1, p = 0.1, list = FALSE)
edx_mut_test <- edx_mut[-test_index,]
edx_mut_temp <- edx_mut[test_index,]

# Make sure userId and movieId in validation set are also in edx_mut_test set
edx_mut_val <- edx_mut_temp %>%
  semi_join(edx_mut_test, by = "movieId") %>%
  semi_join(edx_mut_test, by = "userId")


# Add rows removed from validation set back into edx_mut_test set
removed <- anti_join(edx_mut_temp, edx_mut_val)
edx_mut_test <- rbind(edx_mut_test, removed)

# find optimal lambda

lambdas <- seq(0, 10, 0.5)


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

  mu <- mean(edx_mut_test$rating)

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

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

  predicted_ratings <-
    edx_mut_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_mut_val$rating))
})


qplot(lambdas, rmses)
```



Regularized RMSE using Lambda

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

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

```
## [1] 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 *train* (edx_mut) and *validation* set.

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

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

b_u <- edx_mut %>%
  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)


# calculate rmse
RMSE_Regularized <- RMSE(predicted_ratings, validation$rating)

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

method	RMSE
Simple Average Rating Model	1.0612018
Movie Effects 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 as follows:

method	RMSE
Simple Average Rating Model	1.0612018
Movie Effects Model	0.9439087
Movie + User Effects Model	0.8653488
Regularized Movie + User Effect Model	0.8648177

From the table, we can infer that the RMSEs of each model is an improvement from the previous models, with the regularization model having an RMSE < 0.86490 which is below our target.

Conclusion

We implemented four data models to create movie recommendation systems with different RMSEs. The lowest RMSE(< 0.86490) was achieved with the *Regularized Movie and User Effect Model*. The RMSEs improved as we added more features to the machine learning models.

Future Impovement

Other models like *Matrix Factorization* and Regularization using the *year* and *genres* effects can be used to further improve the RMSE. Also, inclusion of user demographic features can enhance the models.

References

- https://rafalab.github.io/dsbook/introduction-to-machine-learning.html
- 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
- https://grouplens.org/datasets/movielens/10m