Movie Lens Project - Improving Movie Recommendations

Arun Chiriyankandath

22/02/2020

Introduction

Recommendation systems use ratings that users have given to make specific recommendations to other users who may use the products. For example Amazon uses recommendation from other users to predict what a given user will give for a specific item. Similarly Netflix also uses recommendation systems to predict rating for movies from specific users. This project is motivated by the Netflix challenge where users has been asked to create an algorithm which gives 10% improvement than their existing recommendation system. Different models are trained on a training dataset and compared using RMSE method to a validation dataset.

Predicting a movie rating seems have some challenges. There are many different biases in a movie rating.Users used to prefer genres. Some movies are rated depends on the content in them. Some are cranky users who rate movies cranky and give bad ratings for a good movie. In machine learning modeling, these biases are taken in consideration while creating a recommendation system.

Data Wrangling

Getting and Cleaning Data

First step in any data analysis project is to get the data. The Netflix data is not publicly available, but the Group Lens 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 data and clean the code using the below

```
# 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")
if(!require(Metrics))
  install.packages("Metrics", 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)
dl <- "C:\\Users\\91944\\Documents\\R\\R-programming\\Capstone\\ml-10m.zip"
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 edx(Train) and Validation datasets

```
# 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 train set

validation <- temp %>%
     semi_join(edx, by = "movieId") %>%
     semi_join(edx, by = "userId")

# Add rows removed from validation set back into train set

removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

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

Data Exploration

We can see the validation & Train(edx) sets are in Tidy format using the head() function.

```
head(edx)
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
```

```
## 7      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1                Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7       Children|Comedy|Fantasy
```

```
head(validation)
```

```
##   userId movieId rating timestamp
## 1      1     231      5 838983392
## 2      1     480      5 838983653
## 3      1     586      5 838984068
## 4      2     151      3 868246450
## 5      2     858      2 868245645
## 6      2    1544      3 868245920
##                                                  title
## 1                                    Dumb & Dumber (1994)
## 2                                    Jurassic Park (1993)
## 3                                       Home Alone (1990)
## 4                                          Rob Roy (1995)
## 5                                     Godfather, The (1972)
## 6 Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                    genres
## 1                                  Comedy
## 2        Action|Adventure|Sci-Fi|Thriller
## 3                          Children|Comedy
## 4                 Action|Drama|Romance|War
## 5                             Crime|Drama
## 6 Action|Adventure|Horror|Sci-Fi|Thriller
```

Each row represents a rating given by one user u to one movie i.We can see the number of unique users that provide ratings and for how many unique movies they provided them using this code.

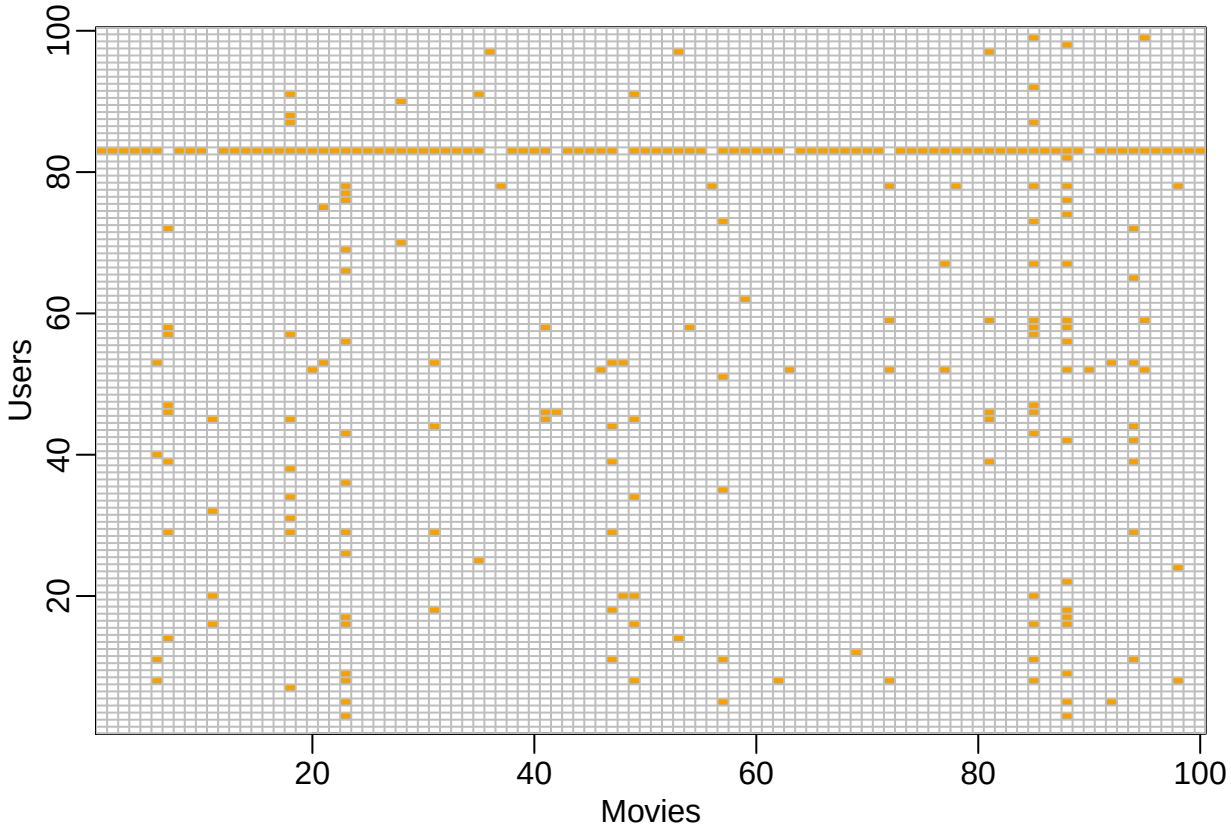
```
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

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

If we multiply those two numbers, we get a number much larger than 10 million.Yet our data table has about less rows than this. This implies that not every user rated every movie.So we can think of this data as a very large matrix with users on the rows and movies on the columns with many empty cells (since all users didn't rate all movies).

The recommendation system which we are creating can be thought of system to fill that empty rows. Below code portraits a 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")
```




```
rm(users)
```

Each outcome y has a different set of predictors. To see this, 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 be used as predictors. But different users rate a different number of movies and different 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. So in essence, the entire matrix can be used as predictors for each cell.

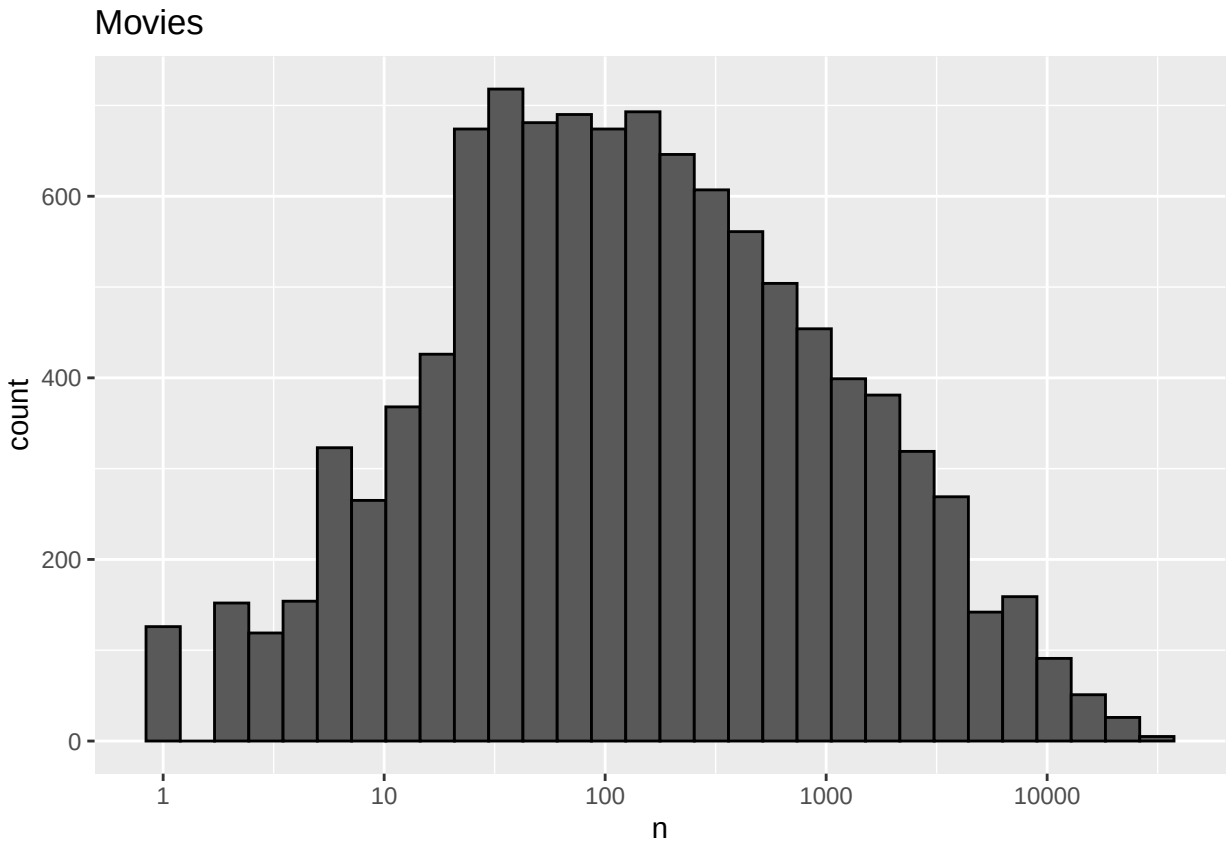
Biases in Data

Lets' see some biases in data

Movie Bias

In this we notice that some movies are rated more than other movies. Given below is the distribution for that:

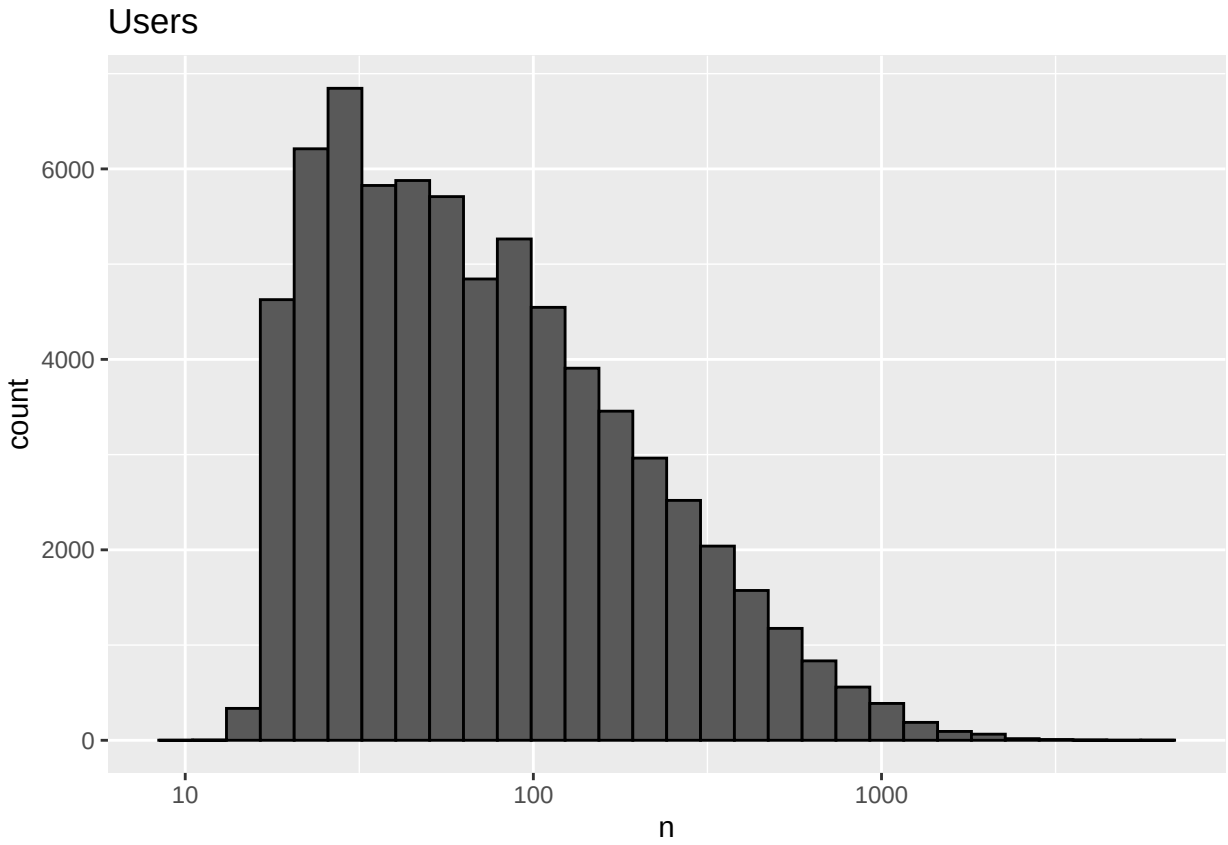
```
edx %>%
  dplyr::count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Movies")
```



User Bias

In this we notice that the ratings are biased on the users. Some users rate lot of movies, but others rate only very few movies.

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



Create Test set and Training Set from Edx dataset

since we cannot use the validation set as the test set, we need to split the edx dataset into test set and train set again. The code is as follows.

```
# Create train set and test set
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx$rating, p=0.2, list = FALSE)
trainSet <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure userId and movieId in test set are also in train set
testSet <- temp %>%
  semi_join(trainSet, by = "movieId") %>%
  semi_join(trainSet, by = "userId")

# Add rows removed from test set back into train set
removed <- anti_join(temp, testSet)
trainSet <- rbind(trainSet, removed)
#rm can be used to remove objects and free space
rm(test_index, temp, removed)
```

Data Analysis

Selecting loss Function

To compare different models or to see how well we're doing compared to some baseline, we need to quantify what it means to do well. We need a loss function. The Netflix challenge used the typical error and thus decided on a winner based on the residual mean squared error on a test set.

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

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

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

We can interpret the residual mean squared error similar to standard deviation. It is the typical error we make when predicting a movie rating. If this number is much larger than one, we're typically missing by one or more stars rating which is not very good.

Let's write 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))
}
```

And now we're ready to build models and compare them to each other using RMSE.

Different Models

Naive approach

Let's start by building the simplest possible recommendation system. We're going to predict the same rating for all movies, regardless of the user and movie. We can use a model based approach A model that assumes the same rating for all movies and users with all the differences explained by random variation would look like this:

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

Here $\epsilon_{u,i}$ is the, independent errors sampled from the same distribution centered at zero, and μ represents 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:

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

```
## [1] 3.512478
```

So that is the average rating of all movies across all users. If we compare all the know ratings in test dataset with $\hat{\mu}$ we obtain the following RMSE:

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

```
## [1] 1.059904
```

This RMSE is quite big. As per the requite of this project, we need to get an RMSE < 0.86490. Now because as we go along we will be comparing different approaches,we're going to create a table that's going to store the results that we obtain as we go along. Let's start by creating a results table with this naive approach:

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

method	RMSE
Naive approach	1.059904

Modeling the movie effects

So let's try a better approach to improve our RMSE. We know from experience that some movies are just generally rated higher than others.We saw that in an earlier plot on the edx dataset. So our intuition that different movies are rated differently is confirmed by data.So we can augment our previous model by adding a term, b_i, to represent the average rating for movie i. The model will look like as below.

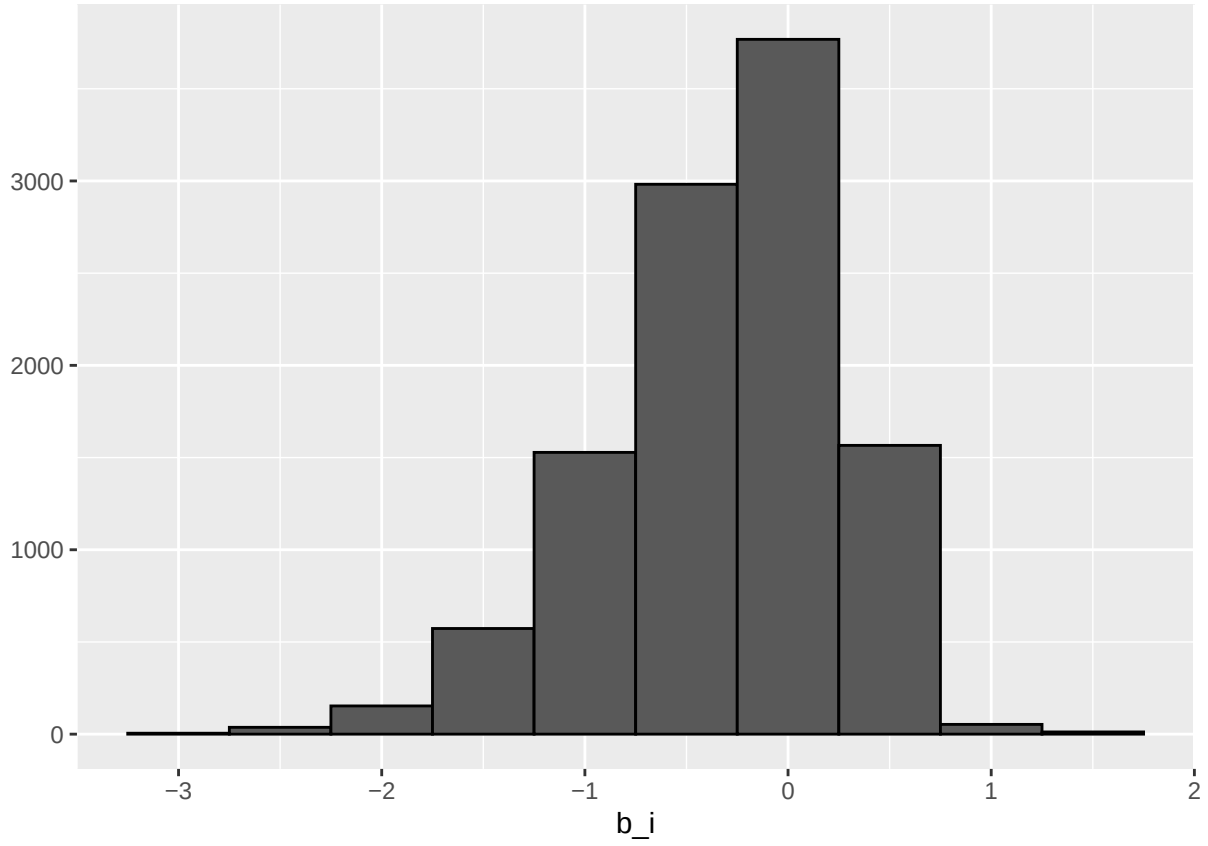
$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

In statistics, we usually call these b's, effects. But in the Netflix challenge papers, they refer to them as *bias*, thus the b in the 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 using the below code:

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

We can see that these estimates vary substantially, not surprisingly. Some movies are good, but others are bad.

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



The overall average which we got is about 3.5.

$\hat{\mu} = 3.5$ $b_i = 1.5$

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

So a b_i of 1.5 implies a perfect five star rating. Now let's see how much our prediction improves once we predict using the model we just fit.

```
# calculate predictions considering movie effect
predicted_ratings <- mu + testSet %>%
     left_join(movie_avgs, by='movieId') %>%
     .$b_i
# calculate rmse after modelling movie effect
model_1_rmse <- RMSE(predicted_ratings, testSet$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	1.0599043
Movie effect model	0.9437429

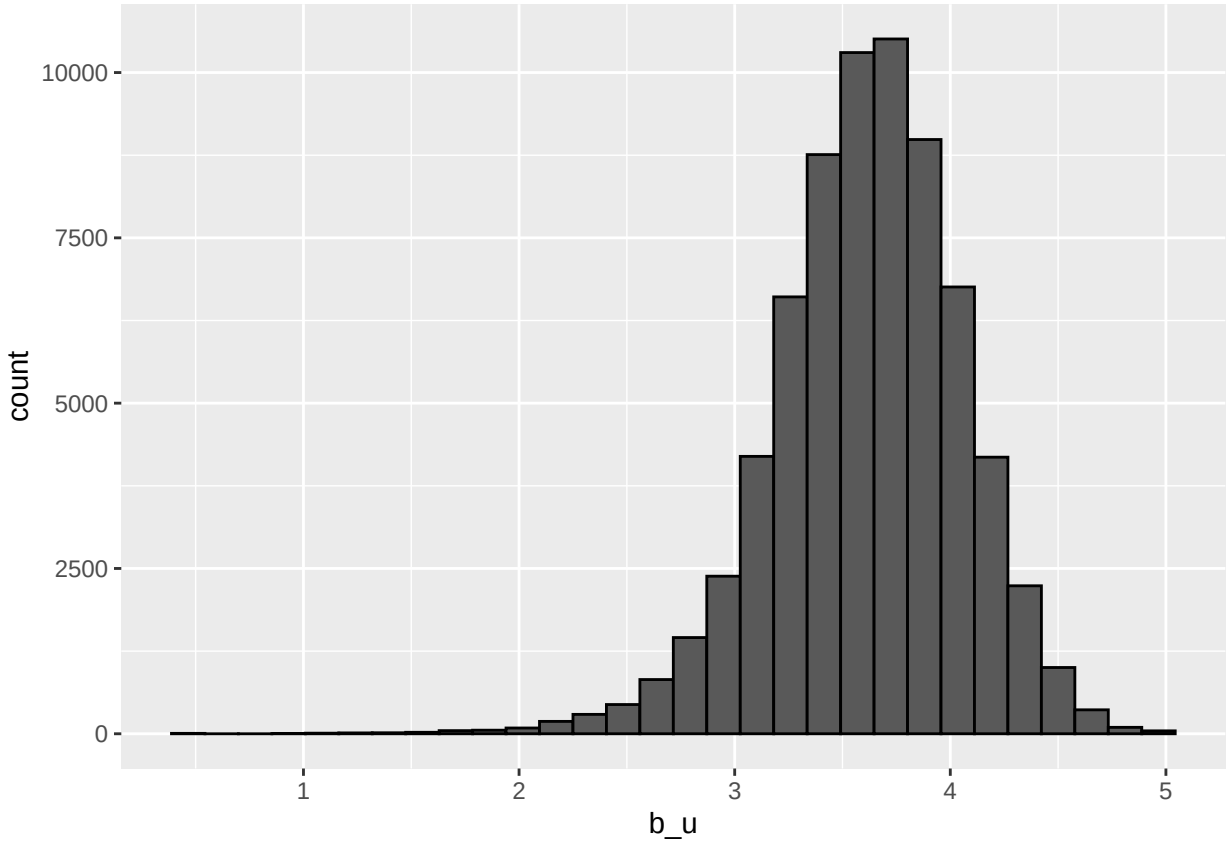
Movie effects model shows a improvement over naive method. But RMSE didn't reached it's goal of less

than 0.86490. So we need to go after other approaches which can bring down the RMSE further below the goal.

Modeling the user effects

Are different users different in terms of how they rate movies? To explore the data, let's compute the average rating for user, u, for those that have rated over 100 movies. We can make a histogram of those values. The graph looks as below.

```
trainSet %>% group_by(userId) %>%
             summarize(b_u = mean(rating)) %>%
             filter(n() >= 100) %>%
             ggplot(aes(b_u)) +
             geom_histogram(bins = 30, color = "black")
```



Note that there is substantial variability across users, as well.Some users are very cranky. And others love every movie they watch, while most of them are in the middle. This implies that a further improvement to our model may be:

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

Here 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 <- trainSet %>%
                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 <- testSet %>%
                        left_join(movie_avgs, by='movieId') %>%
                        left_join(user_avgs, by='userId') %>%
                        mutate(pred = mu + b_i + b_u) %>%
                        .$pred
# calculate rmse after modelling user specific effect in previous model
model_2_rmse <- RMSE(predicted_ratings, testSet$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	1.0599043
Movie effect model	0.9437429
Movie + User effects model	0.8659319

Movie + User effects model shows a improvement over naive method. But RMSE didn't reached it's goal of less than 0.86490. So we need to go after other approaches which can bring down the RMSE further below the goal.

Regularizing movie and user effects

While investigating the cause of less improvement in RMSE, we could see that the top 10 movies and top worst movies are not correct using the movie effect.

TOP 10 BEST MOVIES

```
movie_titles <- edx %>%
     select(movieId, title) %>%
     distinct()
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
     arrange(desc(b_i)) %>%
     select(title, b_i) %>%
     slice(1:10) %>%
     knitr::kable()
```

title	b_i
Hellhounds on My Trail (1999)	1.487522
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.487522
Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	1.487522

title	b_i
Shadows of Forgotten Ancestors (1964)	1.487522
Money (Argent, L') (1983)	1.487522
Fighting Elegy (Kenka erejii) (1966)	1.487522
Sun Alley (Sonnenallee) (1999)	1.487522
Aerial, The (La Antena) (2007)	1.487522
Blue Light, The (Das Blaue Licht) (1932)	1.487522
More (1998)	1.404189
TOP 10 WORST MOVIES	

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

title	b_i
Besotted (2001)	-3.012478
Hi-Line, The (1999)	-3.012478
Accused (Anklaget) (2005)	-3.012478
Confessions of a Superhero (2007)	-3.012478
War of the Worlds 2: The Next Wave (2008)	-3.012478
SuperBabies: Baby Geniuses 2 (2004)	-2.749978
From Justin to Kelly (2003)	-2.667240
Legion of the Dead (2000)	-2.637478
Disaster Movie (2008)	-2.637478
Hip Hop Witch, Da (2000)	-2.603387

So this doesn't make any sense. So let's look at how often they are rated.

TOP BEST MOVIES WITH COUNT OF RATINGS

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

title	b_i	n
Hellhounds on My Trail (1999)	1.487522	1
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.487522	3
Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	1.487522	2
Shadows of Forgotten Ancestors (1964)	1.487522	1
Money (Argent, L') (1983)	1.487522	1
Fighting Elegy (Kenka erejii) (1966)	1.487522	1
Sun Alley (Sonnenallee) (1999)	1.487522	1
Aerial, The (La Antena) (2007)	1.487522	1
Blue Light, The (Das Blaue Licht) (1932)	1.487522	1

title	b_i	n
More (1998)	1.404189	6

TOP WORST MOVIES WITH COUNT OF RATINGS

```
trainSet %>% dplyr::count(movieId) %>%
     left_join(movie_avgs) %>%
     left_join(movie_titles, by="movieId") %>%
     arrange(b_i) %>%
     select(title, b_i, n) %>%
     slice(1:10) %>%
     knitr::kable()
```

title	b_i	n
Besotted (2001)	-3.012478	1
Hi-Line, The (1999)	-3.012478	1
Accused (Anklaget) (2005)	-3.012478	1
Confessions of a Superhero (2007)	-3.012478	1
War of the Worlds 2: The Next Wave (2008)	-3.012478	2
SuperBabies: Baby Geniuses 2 (2004)	-2.749978	40
From Justin to Kelly (2003)	-2.667240	168
Legion of the Dead (2000)	-2.637478	4
Disaster Movie (2008)	-2.637478	28
Hip Hop Witch, Da (2000)	-2.603387	11

So the supposed best and worst movies are rated by very few users in most of the cases.

This happened because with just a few users, we have more uncertainty. Therefore larger estimates of b_i, negative or positive, are more likely when fewer users rate the movies.

These are basically noisy estimates that we should not trust, especially when it comes to prediction. Large errors can increase our residual mean squared error, so we would rather be conservative when we're not sure.

When making predictions, we need one number, one prediction, not an interval.

Regularization

For this, we introduce the concept of *Regularization*. Regularization permits us to penalize large estimates that comes from small sample sizes. The general idea is to add a penalty for large values of b to the sum of squares equations that we minimize.

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

The first term is just the residual sum of 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 using the below formula:

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

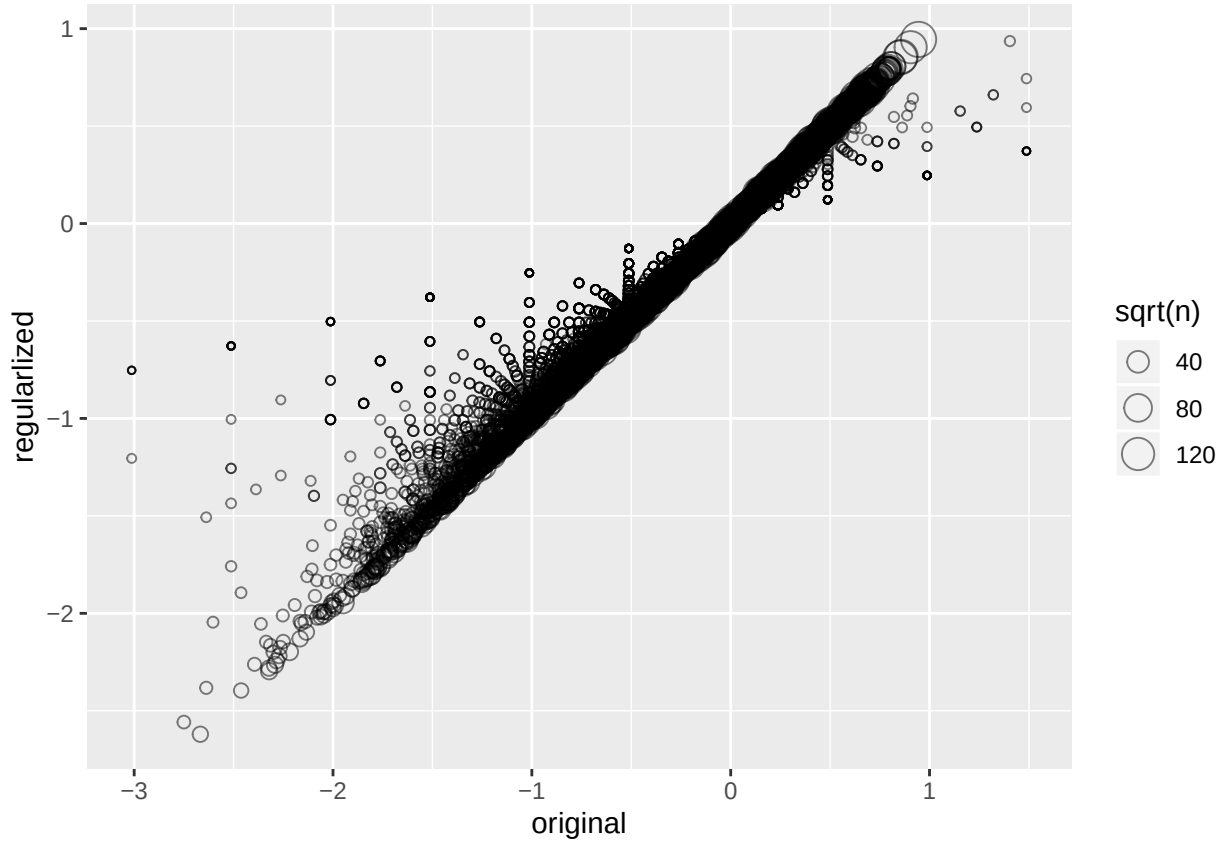
where n_i is the number of ratings b for movie i. This approach will have our desired effect: when our sample size n_i is very large, a case which will give us a stable estimate, then the penalty λ is effectively ignored since $n_i + \lambda \approx n_i$. However, when the n_i is small, then the estimate $\hat{b}_i(\lambda)$ is shrunken towards 0. The larger λ, the more we shrink.

So let's compute the regularized estimates using λ equals to 3.0. Afterwards we can try tuning the λ and see the optimal values of λ.

```
lambda <- 3
mu <- mean(trainSet$rating)
movie_reg_avgs <- trainSet %>%
     group_by(movieId) %>%
     summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
```

To see how the estimates shrink, let's make a plot of the regularized estimate versus the least square estimates with the size of the circle telling us how large ni was.You can see that when n is small, the values are shrinking more towards zero.

```
data_frame(original = movie_avgs$b_i,
           regularlized = movie_reg_avgs$b_i,
           n = movie_reg_avgs$n_i) %>%
     ggplot(aes(original, regularlized, size=sqrt(n))) +
     geom_point(shape=1, alpha=0.5)
```



TOP BEST MOVIES WITH COUNT OF RATINGS AFTER REGULARIZATION

```
trainSet %>%
     dplyr::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) %>%
     knitr::kable()
```

title	b_i	n
Shawshank Redemption, The (1994)	0.9447133	22363
More (1998)	0.9361259	6
Godfather, The (1972)	0.9048175	14107
Usual Suspects, The (1995)	0.8567933	17315
Schindler's List (1993)	0.8514445	18567
Rear Window (1954)	0.8086406	6325
Casablanca (1942)	0.8045806	9027
Double Indemnity (1944)	0.7970539	1711
Third Man, The (1949)	0.7960395	2420
Dark Knight, The (2008)	0.7957394	1900

TOP WORST MOVIES WITH COUNT OF RATINGS AFTER REGULARIZATION

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

title	b_i	n
From Justin to Kelly (2003)	-2.620446	168
SuperBabies: Baby Geniuses 2 (2004)	-2.558119	40
PokÃ©mon Heroes (2003)	-2.396072	109
Disaster Movie (2008)	-2.382238	28
Glitter (2001)	-2.296557	282
Barney's Great Adventure (1998)	-2.281265	168
Gigli (2003)	-2.264313	249
Carnosaur 3: Primal Species (1996)	-2.261785	51
Pokemon 4 Ever (a.k.a. PokÃ©mon 4: The Movie) (2002)	-2.243253	168
Son of the Mask (2005)	-2.218299	128

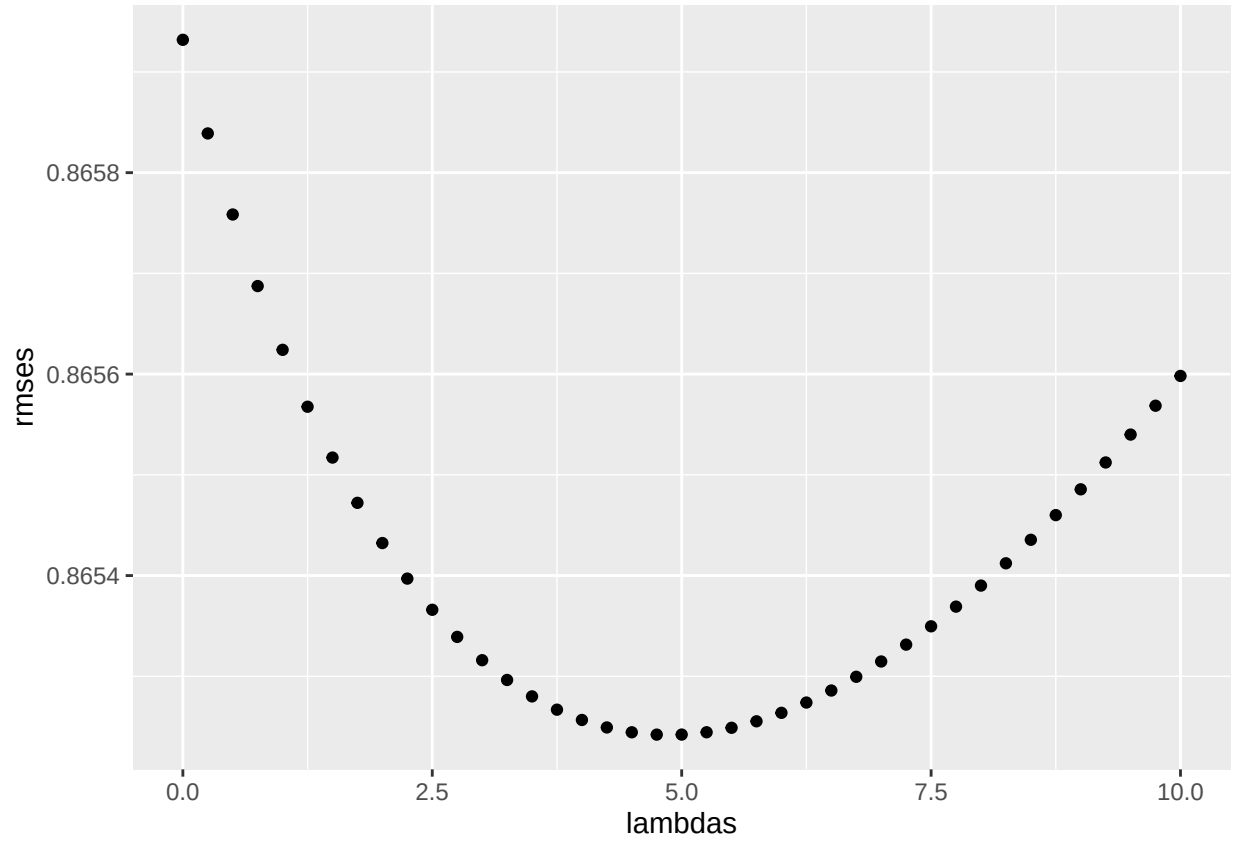
We can see the predictions has been improved a lot for TOP movies and WORST movies.

Note that λ is a tuning parameter in this equation. We can use cross validation to choose it. Also going to add the user effects to improve our predictions.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
     mu <- mean(trainSet$rating)
     b_i <- trainSet %>%
          group_by(movieId) %>%
          summarize(b_i = sum(rating - mu)/(n()+l))
     b_u <- trainSet %>%
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu)/(n()+l))
     predicted_ratings <-
          testSet %>%
          left_join(b_i, by = "movieId") %>%
          left_join(b_u, by = "userId") %>%
          mutate(pred = mu + b_i + b_u) %>%
          .$pred
     return(RMSE(predicted_ratings, testSet$rating))
})
```

When we plot the lambdas against RMSEs generated, we will get the value of lambda when RMSE is small.

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



So from the plot we understood that when λ we are getting the lowest RMSE.

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

```
## [1] 4.75
```

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

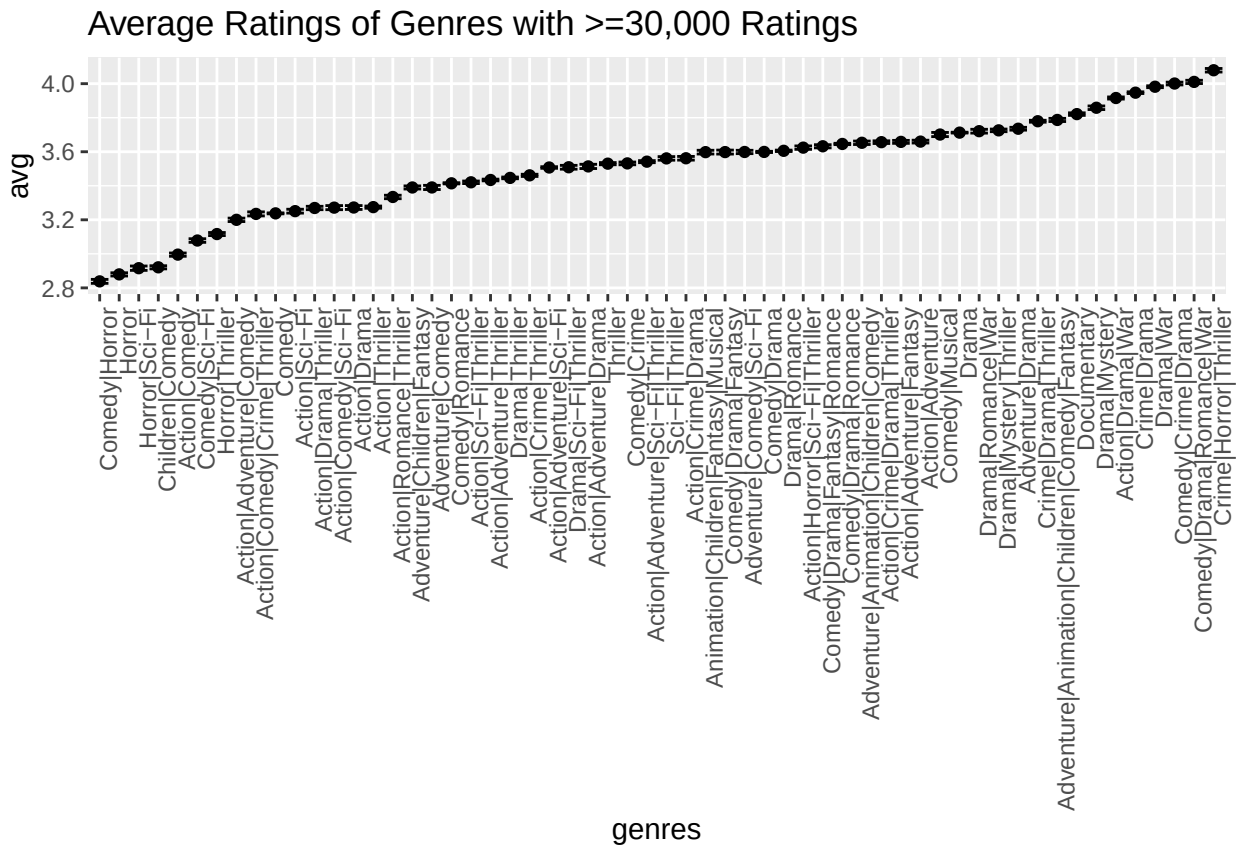
method	RMSE
Naive approach	1.0599043
Movie effect model	0.9437429
Movie + User effects model	0.8659319
Regularized Movie + User Effect Model	0.8652421

Regularized Movie + User effects model shows a improvement over Movie + User effects model method. But RMSE didn't reached it's goal of less than 0.86490. So we need to go after other approaches which can bring down the RMSE further below the goal.

Other Biases

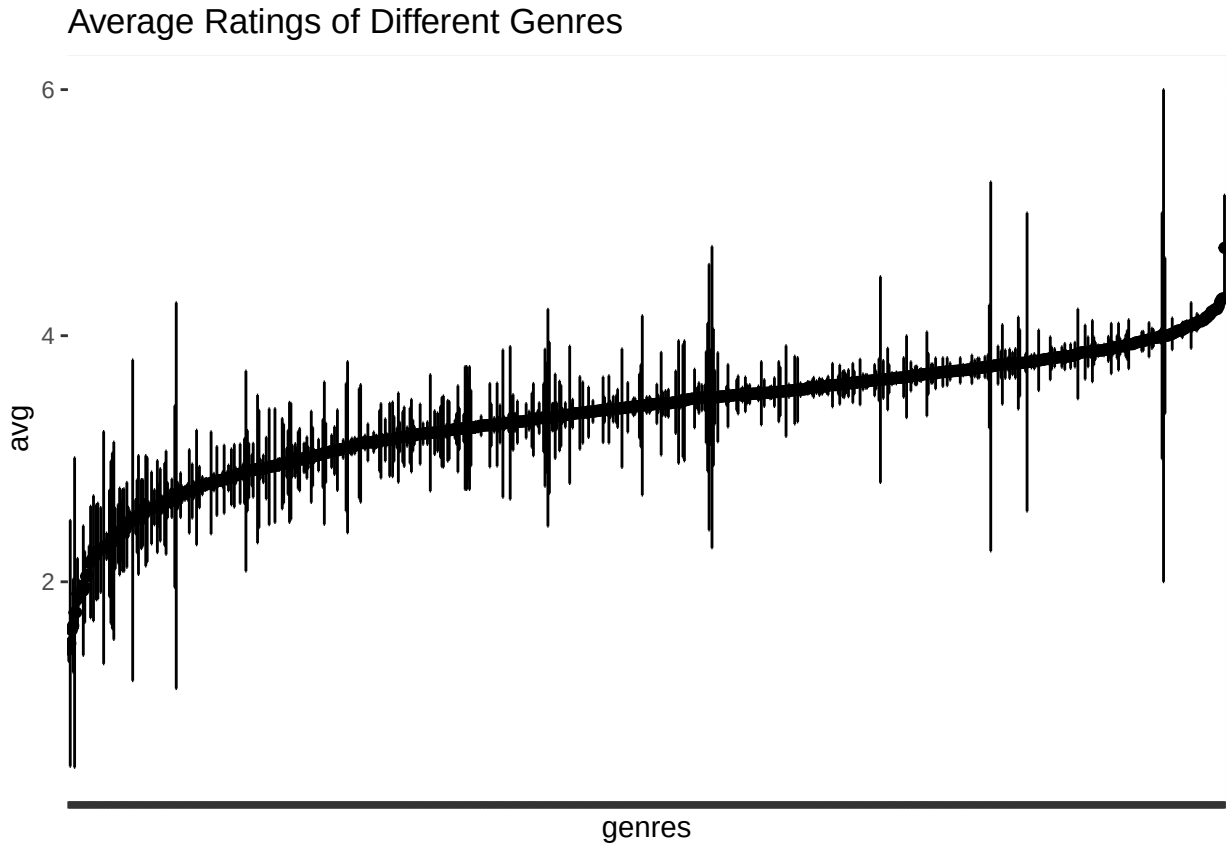
One of the other division we saw in the edx dataset is by *genre*. On plotting error bar plots for average ratings of movies grouped by genres with more than 30,000 ratings, evidence of genre effect b_g is found.

```
#Group ratings by genres and plot the error bar plots for genres with over 30,000 ratings
edx %>% group_by(genres) %>%
  summarize(n=n(),avg=mean(rating),se=sd(rating)/sqrt(n())) %>% #mean and standard errors
  filter(n >= 30000) %>% #Keeping genres with ratings over 30,000
  mutate(genres = reorder(genres, avg)) %>% #order genres by mean ratings
  ggplot(aes(x=genres,y=avg,ymin=avg-2*se,ymax=avg+2*se)) + #lower and upper confidence intervals
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))+
  ggtitle("Average Ratings of Genres with >=30,000 Ratings")
```



if we remove the axis text we can see the average error in the rating for different genres. So the genre effect b_g also we need to take into account.

```
#Group ratings by genres and plot the error bar plots for genres with over 30,000 ratings
edx %>% group_by(genres) %>%
  summarize(n=n(),avg=mean(rating),se=sd(rating)/sqrt(n())) %>% #mean and standard errors
  filter(n>1) %>%
  mutate(genres = reorder(genres, avg)) %>% #order genres by mean ratings
  ggplot(aes(x=genres,y=avg,ymin=avg-2*se,ymax=avg+2*se)) + #lower and upper confidence intervals
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_blank()) + #Removing X-Axis Genre labels
  ggtitle("Average Ratings of Different Genres")
```

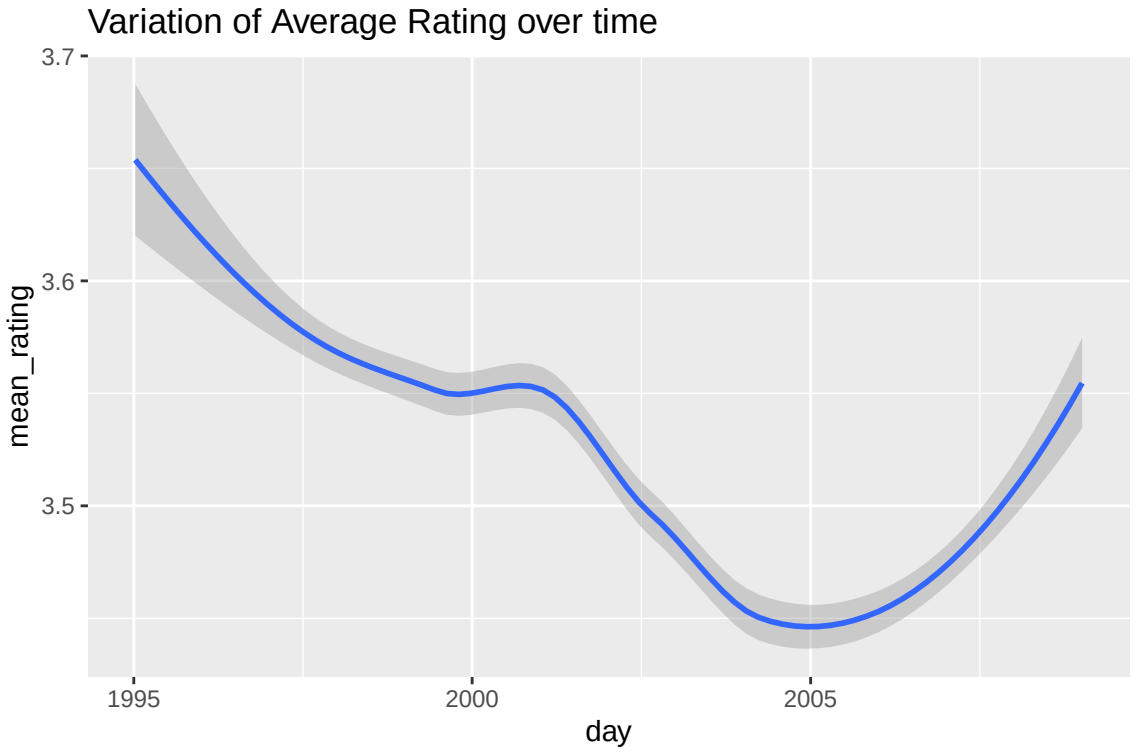


Now the equation for the model without considering the regularization is as below.

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

On further examination, we could see that there is a time effect coming in the rating. The following graph represents this.

```
#Group ratings by date and plot the smoothed conditional mean over the days
edx %>% mutate(time=as_datetime(timestamp)) %>% #Converting epoch time to human readable time
  group_by(day=floor_date(time,"day")) %>% #rounding date-time down to nearest day
  summarise(mean_rating=mean(rating)) %>%
  ggplot(aes(day,mean_rating)) +
  geom_smooth(method='loess', formula = y~x) +
  ggtitle("Variation of Average Rating over time")
```



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

In the plot "Average ratings of Different Genres", some genres have high standard errors. We can filter out genres with high standard errors on their mean ratings to further decrease the RMSE. For genres having high standard errors, it is better to be conservative and not assign a b_g value. The reason is as mentioned earlier we need to have a single value for b_g not a set of values. So we are going to penalize the b_g with high standard error. Since we need to penalize for all the ratings, we need cross validation to find a optimum cutoff which reduces the RMSE.

For determining the cutoff value of the standard error, we use cross-validation.

```
# Using cross validation to determine the optimal standard Error cut off value
ses <- seq(0,1,0.1) #Range of Standard Error values
rmses <- sapply(ses, function(s){
     mu <- mean(trainSet$rating)
     b_i <- trainSet %>%
          group_by(movieId) %>%
          summarize(b_i = sum(rating - mu)/(n()))
     b_u <- trainSet %>%
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu)/(n()))
     b_t <- trainSet %>%
          left_join(b_u, by="userId") %>%
          left_join(b_i, by = "movieId") %>%
          mutate(time=as_datetime(timestamp)) %>%
          group_by(day=floor_date(time,"day")) %>%
          summarise(b_t = sum(rating - b_i - b_u - mu)/(n()))
    b_g <- trainSet %>%
```

```
        left_join(b_u, by="userId") %>%
        left_join(b_i, by = "movieId") %>%
        mutate(day = floor_date(as_datetime(timestamp),"day")) %>%
        left_join(b_t, by="day") %>%
        mutate(b_t=replace_na(b_t,0)) %>%
        group_by(genres) %>%
        summarise(b_g=(sum(rating - b_i - b_u - b_t -mu ))/(n()),se = sd(rating)/sqrt(n())) %>%
        filter(se<=s) # Retaining b_g values that correspond to Standard Error less than or equal to S

# Predicting movie ratings on test set
predicted_ratings <- testSet %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        mutate(day = floor_date(as_datetime(timestamp),"day")) %>%
        left_join(b_t, by="day") %>%
        mutate(b_t=replace_na(b_t,0)) %>%
        left_join(b_g, by="genres") %>%
        mutate(b_g=replace_na(b_g,0)) %>%
        mutate(pred = mu + b_i + b_u + b_t + b_g) %>%
        .$pred

    return(RMSE(predicted_ratings, testSet$rating))
})
```

The optimal value of the standard error after cross validation is as below.

```
# Storing the optimal standard error error value i.e the one which gives lowest RMSE
s_e <- ses[which.min(rmses)]
s_e
```

```
## [1] 0.6
```

So including regularization to each effects, the model equation changes to the following

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

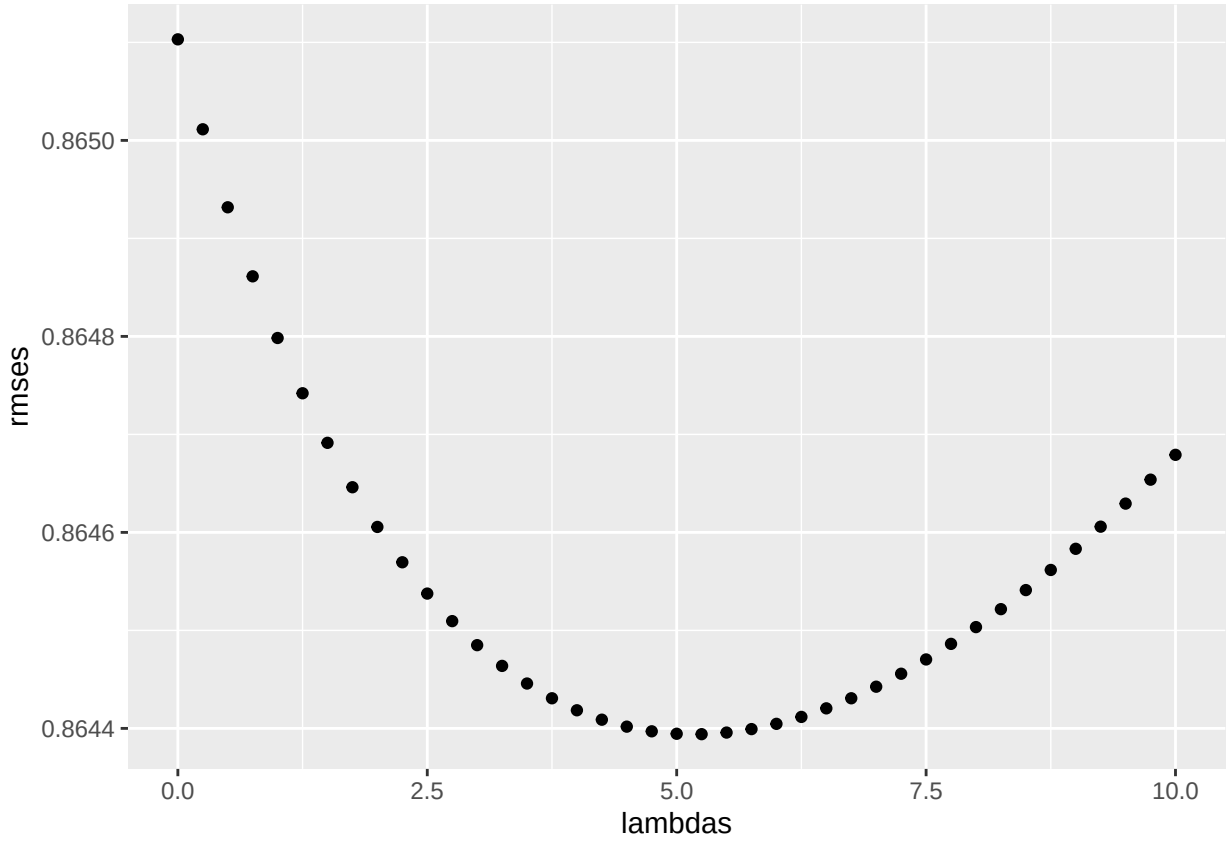
Since λ is a tuning parameter in this equation. We can use cross validation to choose it as mentioned earlier.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
     mu <- mean(trainSet$rating)
     b_i <- trainSet %>%
          group_by(movieId) %>%
          summarize(b_i = sum(rating - mu)/(n()+l))
     b_u <- trainSet %>%
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu)/(n()+l))
     b_t <- trainSet %>%
          left_join(b_u, by="userId") %>%
```

```
          left_join(b_i, by = "movieId") %>%
          mutate(time=as_datetime(timestamp)) %>%
          group_by(day=floor_date(time,"day")) %>%
          summarise(b_t = sum(rating - b_i - b_u - mu)/(n()+l))
     b_g <- trainSet %>%
          left_join(b_u, by="userId") %>%
          left_join(b_i, by = "movieId") %>%
          mutate(day=floor_date(as_datetime(timestamp),"day")) %>%
          left_join(b_t, by="day") %>%
          mutate(b_t=replace_na(b_t,0)) %>%
          group_by(genres) %>%
          summarise(b_g=(sum(rating-b_i-b_u-b_t-mu))/(n()+l),se = sd(rating)/sqrt(n())) %>%
          filter(se<=s_e)
    predicted_ratings <- testSet %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        mutate(day = floor_date(as_datetime(timestamp),"day")) %>%
        left_join(b_t, by="day") %>%
        mutate(b_t=replace_na(b_t,0)) %>% #filling the 'NA' values due to the SE cutoff
        left_join(b_g, by="genres") %>%
        mutate(b_g=replace_na(b_g,0)) %>% #filling the 'NA' values due to the SE cutoff
        mutate(pred = mu + b_i + b_u + b_t + b_g) %>%
        .$pred

    return(RMSE(predicted_ratings, testSet$rating))
})
```

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



So from the plot we understood that when λ we are getting the lowest RMSE.

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

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

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

method	RMSE
Naive approach	1.0599043
Movie effect model	0.9437429
Movie + User effects model	0.8659319
Regularized Movie + User Effect Model	0.8652421
Regularized Movie + User + Time + Genre Effects	0.8643941

Results

So we have reached our final goal of the RMSE below 0.86490 using the model $ReglarisedMovie + User + Time + GenreEffects$. For other models, which we used, we are getting RMSE higher than the required

RMSE. So now lets use the validation dataset and predict the RMSE.

```
# Calculating mean rating
     mu <- mean(trainSet$rating)

# Computing b_i for each movie as mean of residuals
     b_i <- trainSet %>%
          group_by(movieId) %>%
          summarize(b_i = sum(rating - mu)/(n()+lambda))

# Computing b_u for each user as mean of residuals
     b_u <- trainSet %>%
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

# Computing b_t for each day as mean of residuals
     b_t <- trainSet %>%
          left_join(b_u, by="userId") %>%
          left_join(b_i, by = "movieId") %>%
          mutate(time=as_datetime(timestamp)) %>%
          group_by(day=floor_date(time,"day")) %>%
          summarise(b_t = sum(rating - b_i - b_u - mu)/(n()+lambda))

# Computing b_g for each genre as mean of residuals
     b_g <- trainSet %>%
          left_join(b_u, by="userId") %>%
          left_join(b_i, by = "movieId") %>%
          mutate(day=floor_date(as_datetime(timestamp),"day")) %>%
          left_join(b_t, by="day") %>%
          mutate(b_t=replace_na(b_t,0)) %>% #filling the 'NA' values due to the SE cutoff
          group_by(genres) %>%
          summarise(b_g=(sum(rating-b_i-b_u-b_t-mu))/(n()+lambda),se = sd(rating)/sqrt(n())) %>%
          filter(se<=s_e)

# Predicitng the ratings on the validation set
    predicted_ratings <- validation %>%
        left_join(b_i, by = "movieId") %>%
        left_join(b_u, by = "userId") %>%
        mutate(day = floor_date(as_datetime(timestamp),"day")) %>%
        left_join(b_t, by="day") %>%
        mutate(b_t=replace_na(b_t,0)) %>% #filling the 'NA' values due to the SE cutoff
        left_join(b_g, by="genres") %>%
        mutate(b_g=replace_na(b_g,0)) %>% #filling the 'NA' values due to the SE cutoff
        mutate(pred = mu + b_i + b_u + b_t + b_g) %>%
        .$pred
```

The final RMSE is as given below:

```
# RMSE for predictions made on validation set
    RMSE_Valid <- RMSE(predicted_ratings, validation$rating)
    RMSE_Valid
```

```
## [1] 0.8647844
```

Conclusion

In our modeling we have selected the model $ReglarisedMovie + User + Time + GenreEffects$ after going through various models and attained the goal of RMSE < 0.86490. There is even enough scope of improvement using more high end analysis techniques like gradient descent approach, Principal Component Analysis , Ensembles and Singular Value Decomposition.

These approaches require more computer intensive setups and cannot be done / time consuming in regular laptops and desktops.

The learning around the data wrangling, data analysis, data partitioning, model selection, fitting, cross-validating and then final model presentation can be applied to a wide range of technical problems, from forecasting, natural language processing, prediction models, recommender models among many other applications. The skills learned using R Markdown can be applied for creating a number of reports which format in a highly professional manner quickly and simply in a range of formats.

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