Capstone Project - Recommendation System

Renz Marvin S. Asprec

7/4/2021

Executive Summary

This project aims to produce a movie recommendation system, based on the MoiveLens Dataset, through different data analysis strategies via R.

The Movielens data set contains several ratings applied to different movies by many users of the online movie recommender service MovieLens. All users in the data were selected at random and had rated at least 20 movies.

Several approaches were performed in this project. The first approach was a baseline naive approach of just predicting the ratings using the average of all ratings. Then, different biases were investigated to improve the model. The effects of regularization to theses biases were also investigated. To further improve the model, matrix factorization method was also used.

Each approach was evaluated using the root mean squared (RMSE) loss function.

The model which resulted to the least RMSE is the matrix factorization model, with an RMSE of 0.78818.

Introduction

Movielens Data Set

Movielens data set is a collection of 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. This data set was released in January of 2009 by grouplens.

The users were selected at random for inclusion. All users have rated at least 20 movies.

The goal of this project is to create a movie recommendation system based on this data set.

Evaluation Metrics

The resulting models in this project would be evaluated using the loss function, root mean squared error or RMSE.

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

where $y_{u,i}$ is the rating for the user u, and movie i, and $\hat{y}_{u,i}$ is the notation for our predicted rating.

Methodology

Exploratory Data Analsyis

The movielens data was extracted from the grouplens website. The data was partitioned to a 90% training data set, *edx*, and a 10% final hold-out testing data set, *validation.* (Please refer to the Appendix for the code of movielens data extraction)

We can see the *edx* training data in a tidy format below:

```
## # 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
```

From the tibble above, we can see that the data contains the following details in the columns:

- userId - unique identification for a specific user

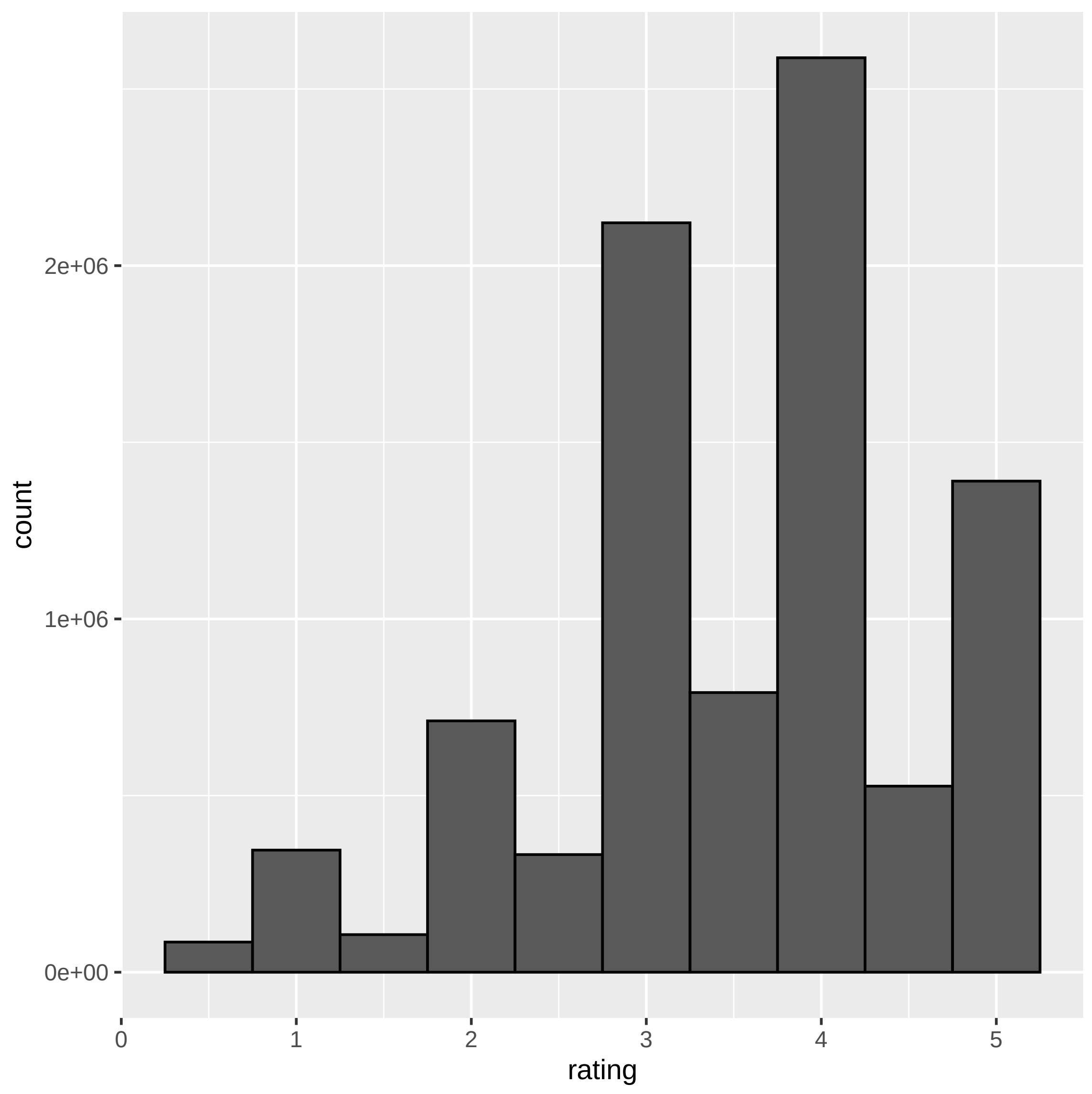
- movieId - unique identification for a specific movie
- rating - rating provided by a specific user for a specific movie. The rating is a 0-5 scale
- timestamp - timestamp of the rating
- title - movie title rated by the specific user
- genres - genres associated with the movie

Our exploration of the data shows that there are 9000055 rows and 6 columns. There are 69878 unique users, 10677 unique movies, and 20 unique genres (including no assigned genres).

Below is the list of the genres:

genres
Comedy
Romance
Action
Crime
Thriller
Drama
Sci-Fi
Adventure
Children
Fantasy
War
Animation
Musical
Western
Mystery
Film-Noir
Horror
Documentary
IMAX
(no genres listed)

The ratings are in a scale of 0-5 stars. Below is a distribution of the ratings:



We can see that the whole star ratings are more common than half star ratings.

Exploration of the data shows that the ratings available are within the years 1995 and 2009

Models

Baseline - Naive Approach

Prediction of ratings based on mean Based on our intuition, we know that the easiest estimate that would reduce the value of the RMSE is the mean. Thus, our first model predicts the rating as the mean of all ratings in our data set. A model that predicts the mean would look like this:

$$\mathrm{Y}_{u,i} = \mu + \epsilon_{u,i}$$

with μ as the average, and $\epsilon_{u,i}$ as the independent errors

Movie Bias

Prediction of rating based on mean, and movie bias This second model assumes that there is a bias for each movies. Some movies are just generally rated higher or lower than other movies. Such model would look like this:

$$\mathrm{Y}_{u,i} = \mu + b_i + \epsilon_{u,i}$$

where b_i is the bias for each movie, i.

User Bias

Prediction of rating based on mean, movie bias, and user bias This third model assumes that there is a bias for each user. Some users generally rate lower or higher compared to other users. Such model would look like this:

$$\mathrm{Y}_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

where b_u is the bias for each user, u.

Genre Bias

Prediction of rating based on mean, movie bias, and genre bias This fourth model assumes that there is a bias for each genre. Some genres are preferred by some users more than the others. Such model would look like this:

$$\mathrm{Y}_{u,i} = \mu + b_i + b_u + b_{u,g} + \epsilon_{u,i}$$

where $b_{u,g}$ is the bias for user, u, for a specific genre, g.

Regularized Movie Bias

Penalize large estimates of b_i formed from small sample sizes The previous linear models did not account for movies that only have a few ratings. These movies with few ratings tend to have more uncertainty. In relation, we want to penalize these movie data that are few in samples, but provide high value of b_is.

We penalize them through the following:

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

where the second term is a penalty term
In order to optimize the equation above, we compute the value of b_i as

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

Here, λ is an optimization parameter.

Regularized User Bias

Penalize large estimates of b_u formed from small sample sizes Similar with the movies, some users have rated only a few number of times. These users with few ratings tend to have more uncertainty. In relation, we want to penalize these user data that are few in samples, but provide high value of b_us.

Regularized Genre Bias

Penalize large estimates of $b_{u,g}$ formed from small sample sizes Similar with previous regularization approaches, we penalize $b_{u,g}s$ that are few in samples, but are high in value.

Matrix Factorization

A popular technique in solving a recommendation system is to use matrix factorization. The idea in matrix factorization is to estimate the whole matrix of rating R_{mxn} by the product of two matrices with lower dimensions, P_{kxm} and Q_{kxn}, such that

$$\mathrm{R} \approx P'Q$$

Let p_u be the u-th column of P, and q_v be the v-th column of Q, then the rating given by the user u on the movie v would be predicted as $p'_u q_v$.

Results

Baseline - Naive Approach

For this model, we will use our training data which we previously defined as, *edx.* We compute the average, mu, of the data.

```
mu
```

```
## [1] 3.5125
```

This average would be our prediction for all the ratings. We can now compute the RMSE of our predicted ratings as compared to the ratings in the , *validation* data set.

```
rmse
```

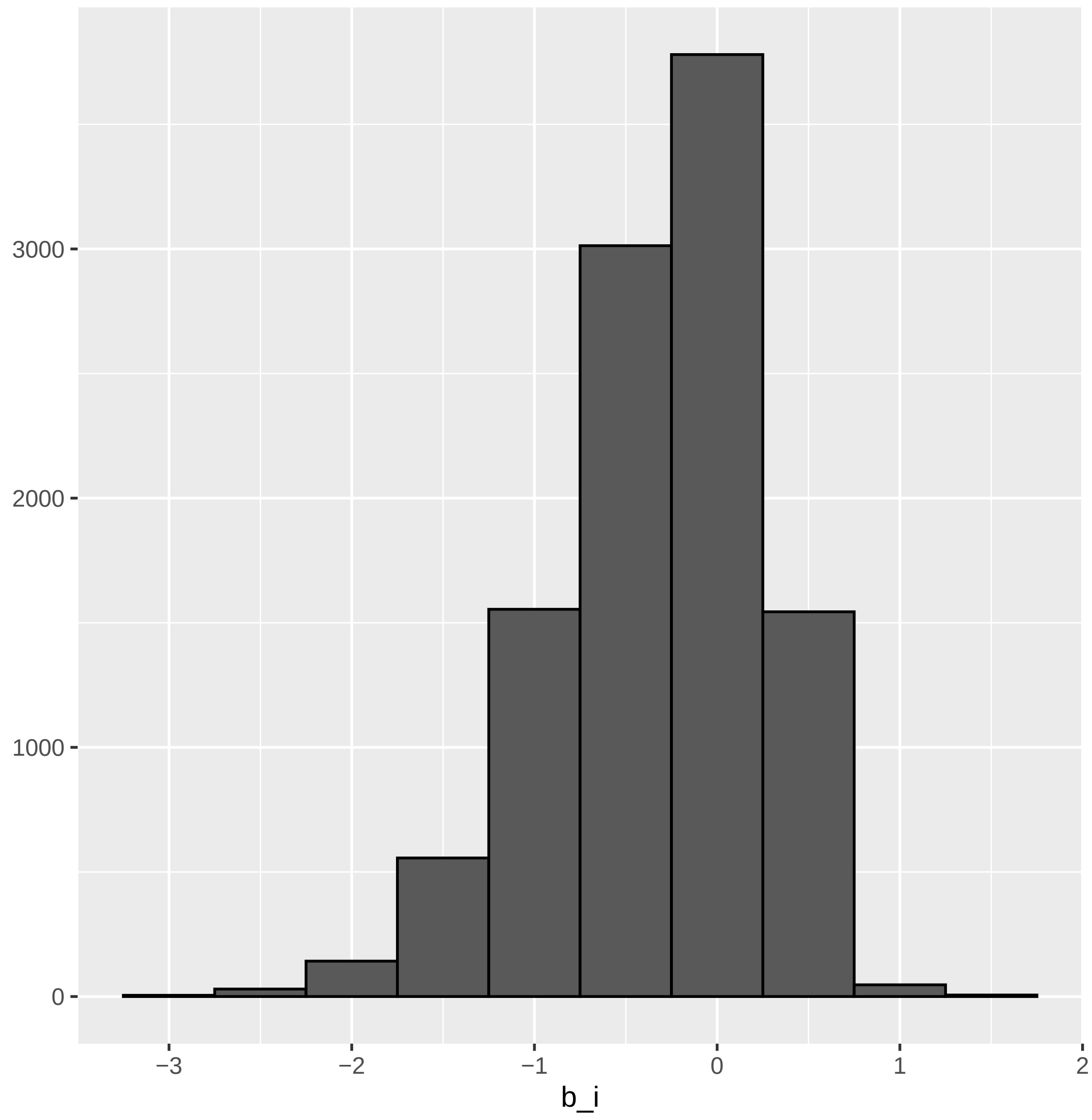
```
## [1] 1.0612
```

We get an RMSE of **1.0612**. This means that our recommendation error would be larger than a star of rating.

model	RMSE
Naive Approach: mean of all ratings	1.0612

Movie Bias

We compute b_i . A plot of b_i is shown below.



The plot of bi shows that indeed, some movies are generally rated higher than others.

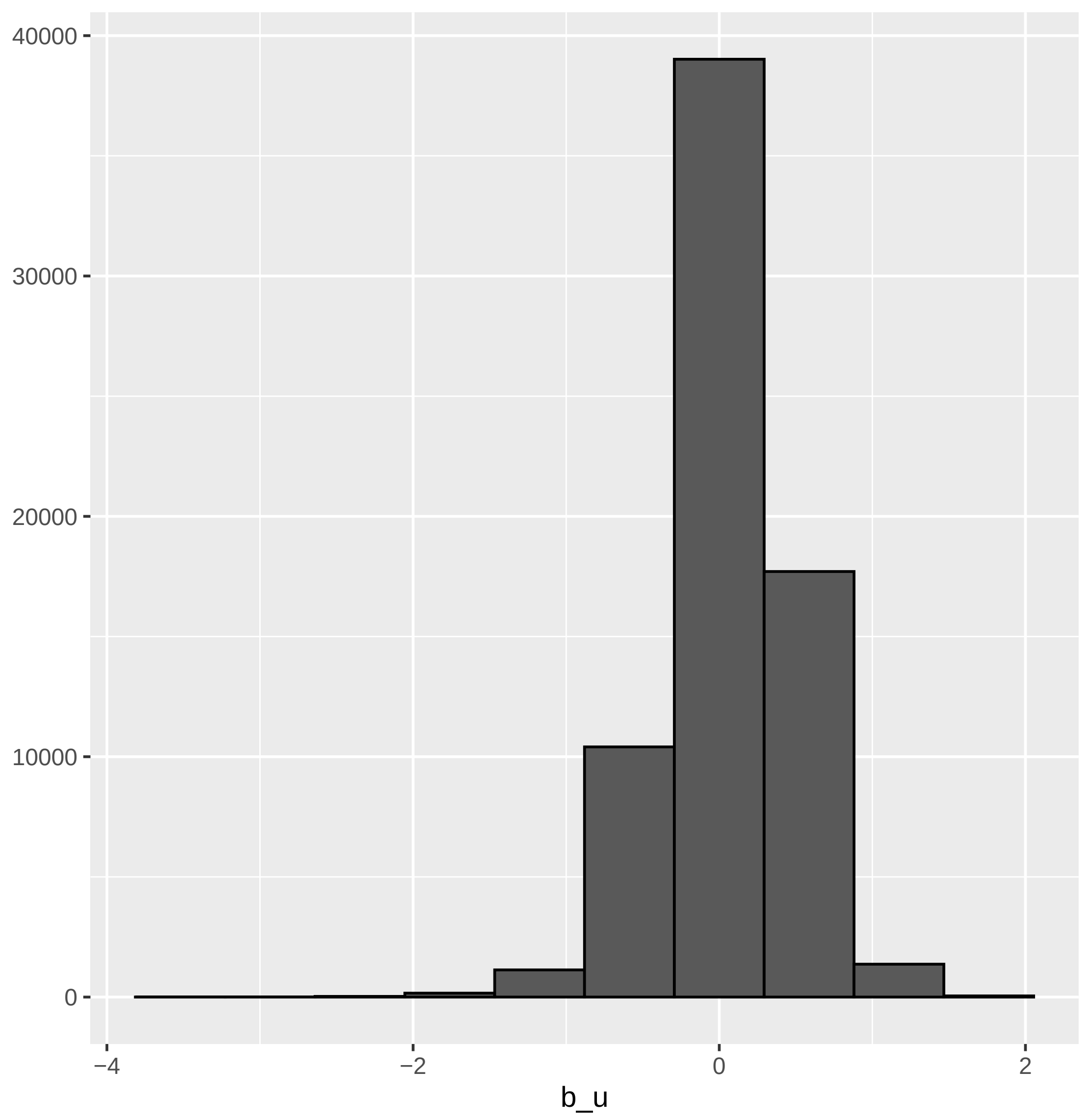
Now, we calculate our predictions using this new model. The resulting RMSE is shown below:

model	RMSE
Naive Approach: mean of all ratings	1.06120
Mean with movie bias	0.94391

The resulting RMSE for this model is **0.94391**. This is an improvement compared to our previous model.

User Bias

We compute b_u. A plot of b_u is shown below.



The plot of b_u shows that some users rate generally lower or higher compared to other users. Although, there's not much variability from 0.

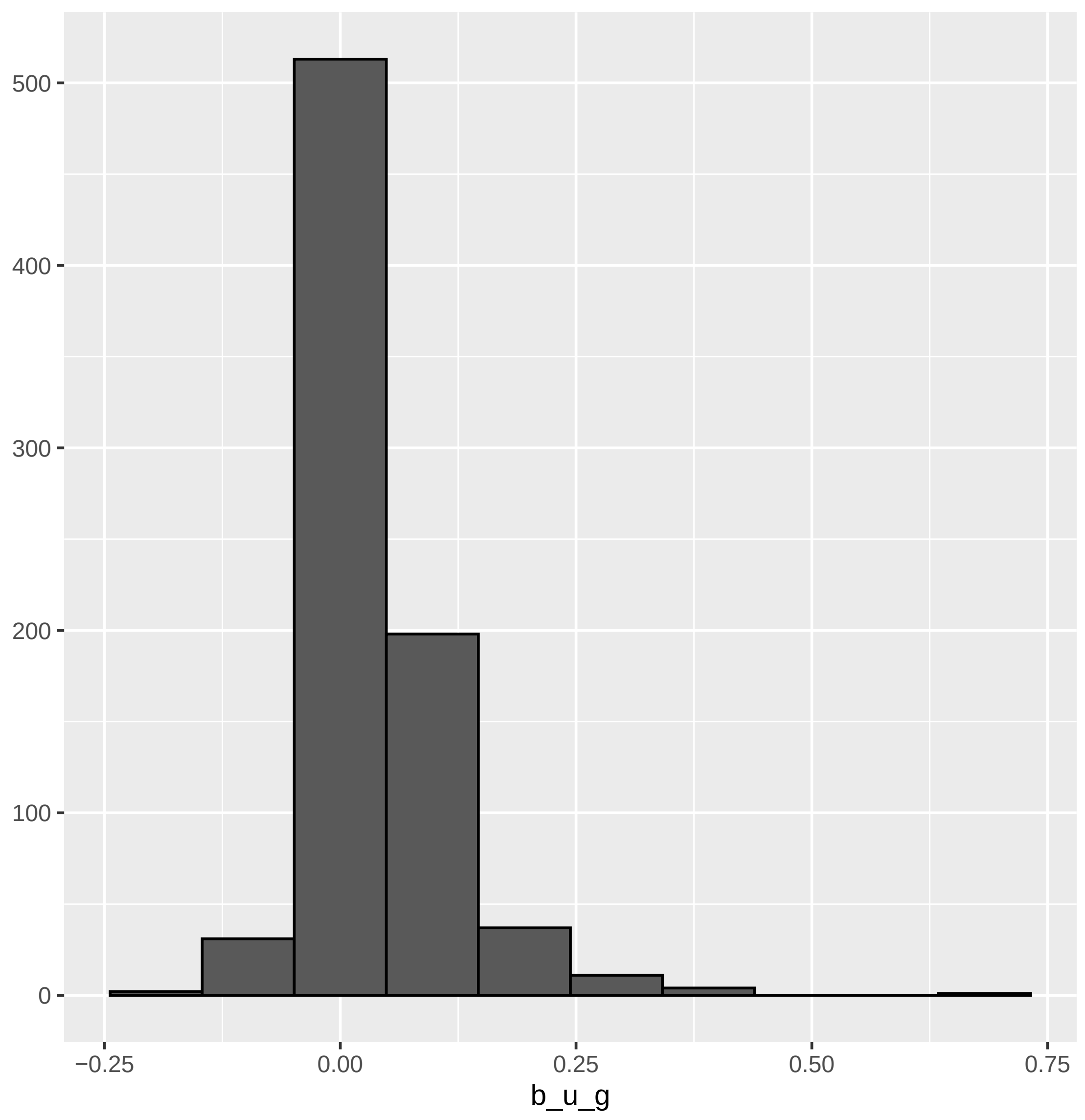
We can now predict our new ratings based on this model. The resulting RMSE is shown below:

model	RMSE
Naive Approach: mean of all ratings	1.06120
Mean with movie bias	0.94391
Mean with movie bias and user bias	0.86535

The resulting RMSE for our new model is **0.86535**. This is an improvement compared to our previous models.

Genre Bias

We compute $b_{u,g}$. The plot of $b_{u,g}$ is shown below:



The plot above showed that, there's not much variability from 0.

We can now predict the ratings for this model. The resulting RMSE is shown below.

model	RMSE
Naive Approach: mean of all ratings	1.06120
Mean with movie bias	0.94391
Mean with movie bias and user bias	0.86535
Mean with movie bias, user bias, and genre bias	0.86495

The resulting RMSE for our new model is **0.86495**. Although an improvement, this model didn't contribute much into lowering the RMSE as we've initially seen from the plot of the variability of the b_us. From this, we can see that the genre bias is not a big contributor in predicting the ratings.

Regularized Movie Bias

Let's investigate the top 10 best and worst movies.

Top 10 movies

title	n
Accused (Anklaget) (2005)	1
Confessions of a Superhero (2007)	1
Hi-Line, The (1999)	1
Besotted (2001)	2
Criminals (1996)	2
War of the Worlds 2: The Next Wave (2008)	2
Hip Hop Witch, Da (2000)	14
Disaster Movie (2008)	32
SuperBabies: Baby Geniuses 2 (2004)	56
From Justin to Kelly (2003)	199

Bottom 10 movies

title	n
Blue Light, The (Das Blaue Licht) (1932)	1
Fighting Elegy (Kenka erejii) (1966)	1
Hellhounds on My Trail (1999)	1
Shadows of Forgotten Ancestors (1964)	1
Sun Alley (Sonnenallee) (1999)	1
Constantine's Sword (2007)	2
Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	2
Human Condition II, The (Ningen no joken II) (1959)	4
Human Condition III, The (Ningen no joken III) (1961)	4
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	4

From the tables above, we can see that the best and worst movies were rated by only a very few users. We expect this since with just a few user rating them, the corresponding ratings have more uncertainty.

It is important to note that in optimizing λ, we can't use the *validation* data set. So, we would partition our *edx* data set to an 80% and 20%, training, and testing sets respectively.

Here are the dimensions of the training, and testing sets.
For the testing set:

```
## [1] 1799967       6
```

For the training set:

```
## [1] 7200088       6
```

We define a range of lambdas that would be used for optimization

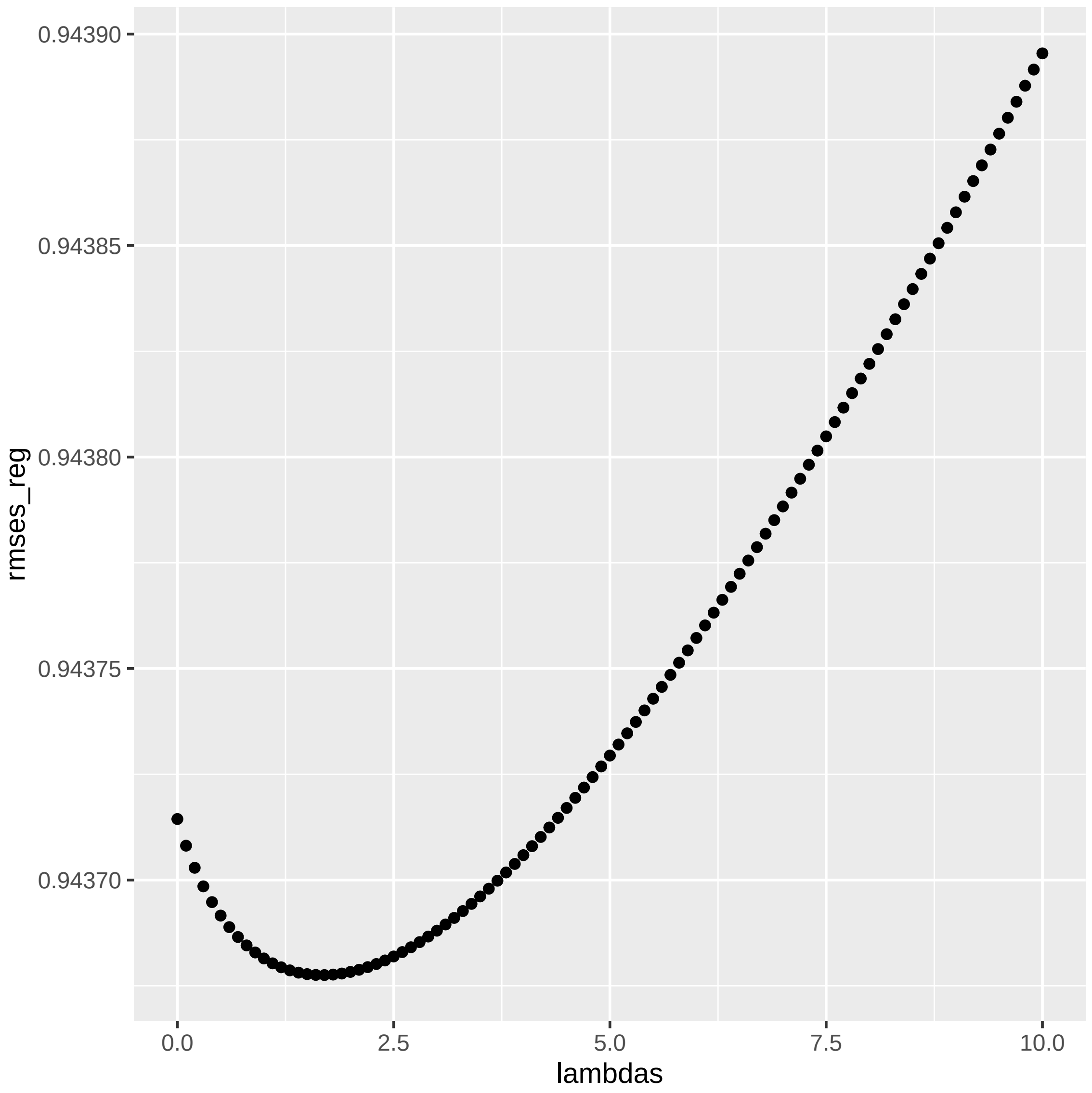
```
lambdas<- seq(0,10,0.1)
```

We compute the average, μ for the train_set

```
mu
```

```
## [1] 3.5126
```

Then, we compute the RMSEs for each lambda. The plot of the resulting RMSEs vs lambdas is shown below.



Obtain lambda which provides the minimum RMSE

```
lambda
```

```
## [1] 1.7
```

Now that we have the optimied λ, we can use this to compute for the RMSE based on the *validation* set.

We recompute the average, μ. Now, for the *edx* data set

```
mu
```

```
## [1] 3.5125
```

We can now compute the regularized movie bias. The resulting RMSE based on this model is shown below:

model	RMSE
Naive Approach: mean of all ratings	1.06120
Mean with movie bias	0.94391
Mean with movie bias and user bias	0.86535
Mean with movie bias, user bias, and genre bias	0.86495
Mean with regularized movie bias	0.94385

The resulting RMSE for this model is **0.94385**. This is an improvement compared to the mean with movie bias model, although by just a very small value. However, it is still less than what we got from the model with movie, user, and genre biases.

Regularized User Bias

We can see below the top 10 users who rated the least number of times

userId	count
62516	10
22170	12
15719	13
50608	13
901	14
1833	14
2476	14
5214	14
9689	14
10364	14

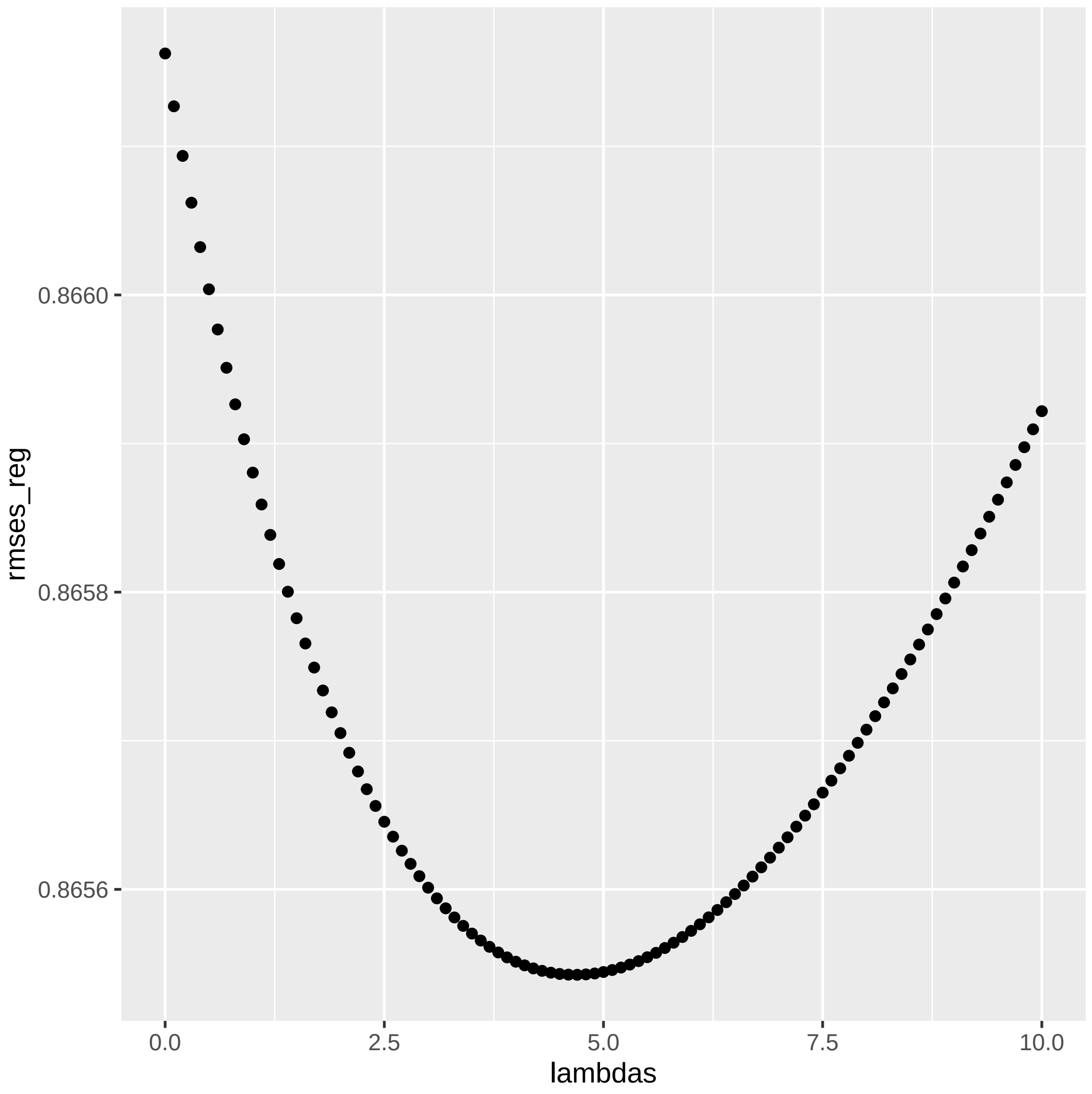
compared to the top 10 users who rated the most number of times

userId	count
59269	6616
67385	6360
14463	4648
68259	4036
27468	4023
19635	3771
3817	3733
63134	3371
58357	3361
27584	3142

Here, we can see that some users rate less compared to other users.

Proceeding with the computation of regularized user bias, we now optimize the penalty term, lambda, through cross-validation.

Here we can see the plot of the RMSEs vs the lambdas



We obtain the lambda that minimizes the RMSE.

```
lambda
```

```
## [1] 4.7
```

Once we optimized our lambda, we can now compute the regularized user bias. The resulting RMSE is shown below:

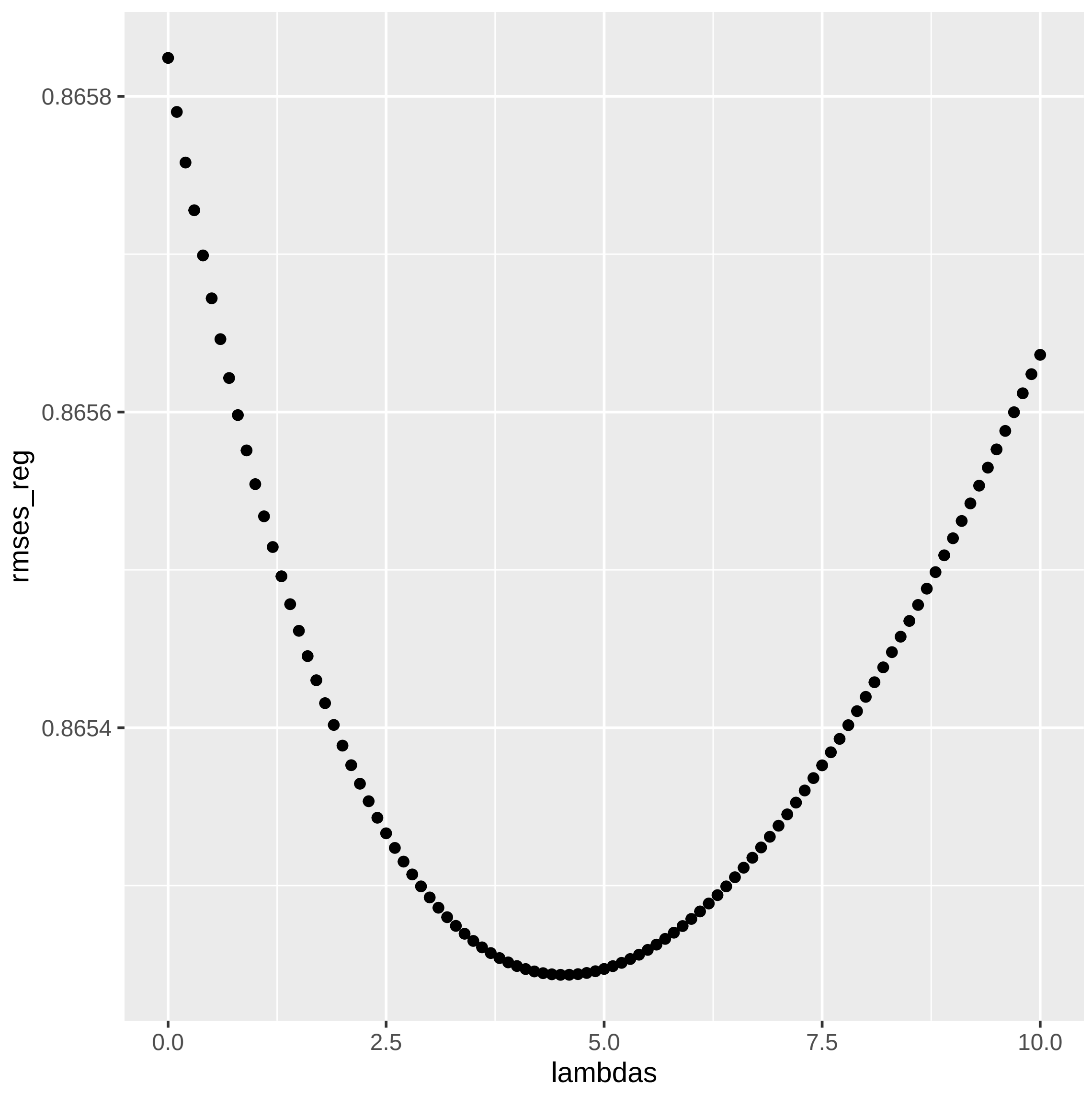
model	RMSE
Naive Approach: mean of all ratings	1.06120
Mean with movie bias	0.94391
Mean with movie bias and user bias	0.86535
Mean with movie bias, user bias, and genre bias	0.86495
Mean with regularized movie bias	0.94385
Mean with regularized movie and user bias	0.86482

The resulting RMSE for our this model is **0.86482**. This is an improvement compared to the mean with regularized movie bias model. However, when compared to the non-regularized model, it did not improve much.

Regularized Genre Bias

Similar, with our previous approaches, we optimize lambda through cross-validation.

The plot below shows the resulting RMSEs vs lambdas:



The lambda which minimizes the RMSE is

```
lambda
```

```
## [1] 4.5
```

We can now proceed with computing the regularized genre bias, based on the *edx* data set. The resulting RMSE is shown below:

model	RMSE
Naive Approach: mean of all ratings	1.06120
Mean with movie bias	0.94391
Mean with movie bias and user bias	0.86535
Mean with movie bias, user bias, and genre bias	0.86495
Mean with regularized movie bias	0.94385

model	RMSE
Mean with regularized movie and user bias	0.86482
Mean and regularized movie, user, and genre bias	0.86445

The resulting RMSE for this model is **0.86445**. This is an improvement compared to the mean with movie bias model, although by just a very small value. Moreover, when compared to the non-regularized model, it did not improve much.

Matrix Factorization

To approach this problem, we can convert the data to a matrix of size equal to the no. of unique users by the no. of unique movies. However, this would be memory intensive. The *recosystem* package in R allows us to perform matrix factorization without crashing our system.

The resulting RMSE for this approach is shown below:

```
## iter      tr_rmse          obj
##    0       0.9666   1.1881e+07
##    1       0.8732   9.8867e+06
##    2       0.8409   9.1932e+06
##    3       0.8199   8.7814e+06
##    4       0.8056   8.5145e+06
##    5       0.7954   8.3328e+06
##    6       0.7874   8.1933e+06
##    7       0.7806   8.0843e+06
##    8       0.7748   7.9955e+06
##    9       0.7699   7.9210e+06
##   10       0.7658   7.8590e+06
##   11       0.7621   7.8075e+06
##   12       0.7588   7.7625e+06
##   13       0.7560   7.7235e+06
##   14       0.7534   7.6886e+06
##   15       0.7512   7.6575e+06
##   16       0.7491   7.6315e+06
##   17       0.7473   7.6101e+06
##   18       0.7456   7.5868e+06
##   19       0.7441   7.5684e+06
```

model	RMSE
Naive Approach: mean of all ratings	1.06120
Mean with movie bias	0.94391
Mean with movie bias and user bias	0.86535
Mean with movie bias, user bias, and genre bias	0.86495
Mean with regularized movie bias	0.94385
Mean with regularized movie and user bias	0.86482
Mean and regularized movie, user, and genre bias	0.86445
Matrix Factorization	0.78818

The resulting RMSE for this model is **0.78818**. This is a significant improvement from our previous models.

Conclusion

For this project, we've explored 8 different models in total. Our first approach is the baseline approach of just predicting the mean of all the ratings. We then investigated the effect of different biases such as movie, user, and genre biases. We saw that taking these biases into account improved our RMSE. In relation, we saw that the movie and user biases contributed more to minimizing the RMSE compared to the genre bias.

We then investigated the effect of regularization to our models. Regularization generally improved the resulting RMSE. However, it did not improve it by much.

Lastly, we used matrix factorization in predicting the ratings. Among the models, this is the most computer-intensive, but also significantly reduced the resulting RMSE.

The model which produced the least RMSE is the **matrix factorization** model with an RMSE of **0.78818**.

Limitations

The recommendation system methods investigated here in this project does not account for new users. New users have no existing information to use as basis for recommendation. Moreover, every time a new movie or user is added, the data should be rerun to take these new data into account.

Appendix

Code provided by edx:

```
##########################################################
# Create edx set, validation set (final hold-out test 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")

library(tidyverse)
library(caret)
library(data.table)

# 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")

# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(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)`
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)
```

Code by user:

```r
# install packages and load libraries
if(!require(knitr)) install.packages("knitr", repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(tinytex)) install.packages("tinytex", repos = "http://cran.us.r-project.org")
if(!require(recosystem)) install.packages("recosystem", repos = "http://cran.us.r-project.org")

library(knitr)
library(lubridate)
library(tinytex)
library(recosystem)

# options
options(digits=5)

# Initial exploration of the data----
as_tibble(edx)

# no of unique users
n_users<- edx %>% select(userId) %>%
  distinct() %>%
  summarize(n=n())

# list of genres
l_genres<- edx %>% separate_rows(genres, sep="\\|") %>%
  select(genres) %>%
  distinct()
l_genres %>% kable()

# no of genres
l_genres %>% summarize(n=n())

# distribution of ratings
edx %>%
  select(rating) %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth=0.5, color="black")

# convert timestamp to date format
edx_date<- edx %>%
  mutate(date=as_datetime(timestamp)) %>%
  select(-timestamp)

# year range of the ratings
year(range(edx_date$date))

# create evaluation metrics, RMSE
RMSE<- function(rating_true, rating_predicted){
  sqrt(mean((rating_true-rating_predicted)^2))
}
# Naive Approach - Prediction of rating based on the average of all ratings ----------------
# compute the average of all ratings
mu<- mean(edx$rating)

# perform rating prediction based on the average of all ratings
rating_predicted<- mu

# resulting rmse
rmse<- RMSE(validation$rating,rating_predicted)

# table of rmses
rmses_results<- data.frame(model= "Naive Approach: mean of all ratings", RMSE= rmse)


# Movie bias - Prediction of rating based on the average ratings and a movie bias, b_i-----
# compute the average of all ratings
mu<- mean(edx$rating)

# compute b_i by averaging the residuals for each movie
movie_averages<- edx %>%
  group_by(movieId) %>%
  summarize(b_i= mean(rating- mu))

# plot of b_i
qplot(movie_averages$b_i, bins=10, color=I("black"), xlab="b_i")

# predict rating taking into account the movie bias, b_i
rating_predicted<- validation %>%
  left_join(movie_averages, by="movieId") %>%
  mutate(prediction= mu + b_i) %>%
  pull(prediction)

# compute the RMSE taking into account the movie bias, b_i
rmse<- RMSE(validation$rating, rating_predicted)
rmses_results<- bind_rows(rmses_results,
                          data.frame(model="Mean with movie bias", RMSE= rmse))

# User-bias - Prediction of rating based on the average ratings, a movie bias, b_i, and a user bias, b_

# compute average for all ratings
mu<- mean(edx$rating)

# compute b_u by averaging the residuals(including the bias on movies) for each user
user_averages<- edx %>%
  left_join(movie_averages, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating-mu-b_i))

# plot of b_u
qplot(user_averages$b_u, bins=10, color=I("black"), xlab="b_u")

# predict ratings taking into account movie bias, b_i, and user bias, b_u
```

```
rating_predicted<- validation %>%
  left_join(movie_averages, by="movieId") %>%
  left_join(user_averages, by="userId") %>%
  summarize(prediction = mu+b_i+b_u) %>%
  pull(prediction)

# compute RMSE
rmse<- RMSE(validation$rating, rating_predicted)
rmses_results<- bind_rows(rmses_results,
                          data.frame(model="Mean with movie bias and user bias", RMSE=rmse))

# Genre-bias Approach - Prediction of rating based on a movie-bias, b_i, user-bias, b_u, and genre-bias

# compute average for all ratings
mu<- mean(edx$rating)

# compute b_u_g by averaging the residuals (including the movie-bias, and user-bias)
genre_averages<- edx %>%
  left_join(movie_averages, by="movieId") %>%
  left_join(user_averages, by="userId") %>%
  group_by(genres) %>%
  summarize(b_u_g = mean(rating-mu-b_i-b_u))

# plot of b_u_g
qplot(genre_averages$b_u_g, bins=10, color=I("black"), xlab="b_u_g")

# predict ratings taking into account the movie bias, b_i, user bias, b_u, and genre bias, b_u_g
rating_predicted<- validation %>%
  left_join(movie_averages, by="movieId") %>%
  left_join(user_averages, by="userId") %>%
  left_join(genre_averages, by="genres") %>%
  summarize(prediction = mu+ b_i+b_u+b_u_g) %>%
  pull(prediction)

# compute RMSE taking
rmse<- RMSE(validation$rating, rating_predicted)
rmses_results<- bind_rows(rmses_results, data.frame(model="Mean with movie bias, user bias, and genre bi

# Regularized Movie bias Approach - Penalize large estimates of b_i formed from small sample sizes-----

# mean of ratings from edx
mu<- mean(edx$rating)

# output movie titles
movie_titles <- edx %>%
  select(movieId, title) %>%
  distinct()

# output top and bottom 10 movies according to estimate
top_movies<- movie_averages %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  slice(1:10)  %>%
  pull(title)

bottom_movies<- movie_averages %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  slice(1:10)  %>%
  pull(title)

# investigate number of ratings for the movies in top and bottom 10
top_10_count<- edx %>%
  filter(title %in% top_movies) %>%
  group_by(title) %>%
  summarize(n=n()) %>%
  arrange(n)

bottom_10_count<- edx %>% filter(title %in% bottom_movies) %>%
  group_by(title) %>%
  summarize(n=n()) %>%
  arrange(n)

# output top and bottom 10 movies
top_10_count %>% kable()
bottom_10_count %>% kable()

# optimize the penalty, lambda (using only the training set, edx)
# cross-validation using 20% as test data. Create a train set and a test set based on the edx data, for

ind<- createDataPartition(edx$rating, times=1, p=0.2, list=FALSE)
temp<- edx[ind]
train_set<- edx[-ind]

# make sure that all movieId, and userId in the test_set are in the train_set
test_set <- temp %>%
  semi_join(train_set, by="movieId") %>%
  semi_join(train_set, by="userId")

# Add rows removed from test_set back into train_set set
removed<- anti_join(temp, test_set)
train_set<- bind_rows(train_set,removed)

# Dimensions of training and testing set
dim(test_set)
dim(train_set)

# range of lambdas used for optimization
lambdas<- seq(0,10,0.1)

# compute average, mu, for the train_set
mu<- mean(train_set$rating)

# function to compute for the rmses for the assumed lambdas
rmses_reg<- sapply(lambdas, function(l){
  # solves for the regularized_movie_averages
  movie_averages_reg<- train_set %>%
  group_by(movieId) %>%
  summarize(b_i_reg = sum(rating-mu)/(n()+l), n_i=n())

  # predict rating based on b_i_reg
  rating_predicted_reg<- test_set %>%
    left_join(movie_averages_reg, by="movieId") %>%
    summarize(prediction = mu+b_i_reg) %>%
    pull(prediction)

  # compute for rmse
  RMSE(test_set$rating, rating_predicted_reg)
})

# get lambda which provides the minimum rmse
lambda<- lambdas[which.min(rmses_reg)]

# plot rmse vs lambda
qplot(lambdas,rmses_reg)

#recompute average, mu, for the edx set
mu<- mean(edx$rating)

# compute regularized movie bias, b_i_reg
movie_averages_reg<- edx %>%
  group_by(movieId) %>%
  summarize(b_i_reg= sum(rating-mu)/(n()+lambda), n_i=n())

# predict rating based on b_i_reg
rating_predicted_reg<- validation %>%
  left_join(movie_averages_reg, by="movieId") %>%
  summarize(prediction = mu+b_i_reg) %>%
  pull(prediction)

# compute for RMSE
rmse<- RMSE(validation$rating, rating_predicted_reg)
rmses_results<- bind_rows(rmses_results,
                          data.frame(model= "Mean with regularized movie bias", RMSE=rmse))
rmses_results %>% kable()

# Regularized User bias approach - Penalize large estimates of b_i and b_u formed from small sample siz

# show the bottom 10 users based on number of ratings
edx %>% group_by(userId) %>%
  summarize(count=n()) %>%
  arrange(count) %>%
  slice(1:10) %>%
  kable()

# show top 10 users based on number of ratings
edx %>% group_by(userId) %>%
  summarize(count=n()) %>%
  arrange(desc(count)) %>%
  slice(1:10) %>%
  kable()

# optimize the penalty, lambda (using only the training set, edx)

# range of lambdas used for optimization
lambdas<- seq(0,10,0.1)

# compute for regularized b_u
mu<- mean(train_set$rating)

# compute RMSEs
rmses_reg<- sapply(lambdas, function(l){

  # compute for b_i_reg
  movie_averages_reg<- train_set %>%
    group_by(movieId) %>%
    summarize(b_i_reg = sum(rating-mu)/(n()+l), n_i=n())

  # compute for b_u_reg
  user_averages_reg<- train_set %>%
    left_join(movie_averages_reg, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u_reg = sum(rating-mu-b_i_reg)/(n()+l))

  # predict ratings
  rating_predicted_reg<- test_set %>%
    left_join(movie_averages_reg, by="movieId") %>%
    left_join(user_averages_reg, by="userId") %>%
    summarize(prediction = mu+b_i_reg+b_u_reg) %>%
    pull(prediction)

  # compute RMSE
  RMSE(test_set$rating, rating_predicted_reg)
})

# plot lambdas vs rmses
qplot(lambdas,rmses_reg)

# obtain lambda that minimizes the rmses of the assigned test_set within the edx data
lambda<- lambdas[which.min(rmses_reg)]

# compute the RMSE of the validation set based on the optimized lambda
mu<- mean(edx$rating)

# compute for b_i_reg
movie_averages_reg<- edx %>%
  group_by(movieId) %>%
  summarize(b_i_reg = sum(rating-mu)/(n()+lambda), n_i=n())

# compute for b_u_reg
user_averages_reg<- edx %>%
```

```r
  left_join(movie_averages_reg, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u_reg = sum(rating-mu-b_i_reg)/(n()+lambda))

# predict rating
rating_predicted_reg<- validation %>%
  left_join(movie_averages_reg, by="movieId") %>%
  left_join(user_averages_reg, by="userId") %>%
  summarize(prediction = mu+b_i_reg+b_u_reg) %>%
  pull(prediction)

# compute for RMSE based on the edx and validation data set
rmse<- RMSE(validation$rating,rating_predicted_reg)
rmses_results<- bind_rows(rmses_results,
                          data.frame(model="Mean with regularized movie and user bias", RMSE=rmse))
rmses_results %>% kable()

# Regularized genre-based approach - Penalize large estimates of b_i, b_u, and b_u_g formed from small 

# use training set and test set obtained from the edx data set to optimize lambda

# range of lambdas used for optimization
lambdas<- seq(0,10,0.1)

mu<- mean(train_set$rating)
rmses_reg<- sapply(lambdas, function(l){
  # compute for b_i
  movie_averages_reg<- train_set %>%
    group_by(movieId) %>%
    summarize(b_i_reg= sum(rating-mu)/(n()+l), n_i=n())

  # compute for b_u
  user_averages_reg<- train_set %>%
    left_join(movie_averages_reg, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u_reg = sum(rating-mu-b_i_reg)/(n()+l))

  # compute for b_u_g
  genre_averages_reg<- train_set %>%
    left_join(movie_averages_reg, by="movieId") %>%
    left_join(user_averages_reg, by="userId") %>%
    group_by(genres) %>%
    summarize(b_u_g_reg= sum(rating-mu-b_i_reg-b_u_reg)/(n()+l))

  # prediction based on test_set obtained from the edx data set
  rating_predicted_reg<- test_set %>%
    left_join(movie_averages_reg, by="movieId") %>%
    left_join(user_averages_reg, by="userId") %>%
    left_join(genre_averages_reg, by="genres") %>%
    summarize(prediction = mu+b_i_reg+b_u_reg+b_u_g_reg) %>%
    pull(prediction)

  # compute for RMSE
  RMSE(test_set$rating, rating_predicted_reg)
})

# plot of the resulting RMSEs vs lambdas
qplot(lambdas, rmses_reg)

# obtain lambda which optimizes the RMSES
lambda<- lambdas[which.min(rmses_reg)]

# compute the RMSE of the validation set based on the optimized lambda
mu<- mean(edx$rating)

# compute for b_i
movie_averages_reg<- edx %>%
  group_by(movieId) %>%
  summarize(b_i_reg = sum(rating-mu)/(n()+lambda), n_i=n())

# compute for b_u
user_averages_reg<- edx %>%
  left_join(movie_averages_reg, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u_reg = sum(rating-mu-b_i_reg)/(n()+lambda))

# compute for b_u_g
genre_averages_reg<- edx %>%
  left_join(movie_averages_reg, by="movieId") %>%
  left_join(user_averages_reg, by="userId") %>%
  group_by(genres) %>%
  summarize(b_u_g_reg = sum(rating-mu-b_i_reg-b_u_reg)/(n()+lambda))

# predict rating based on the edx and validation data set
rating_predicted_reg<- validation %>%
  left_join(movie_averages_reg, by="movieId") %>%
  left_join(user_averages_reg, by="userId") %>%
  left_join(genre_averages_reg, by="genres") %>%
  summarize(prediction= mu+b_i_reg+b_u_reg+b_u_g_reg) %>%
  pull(prediction)

# compute for RMSE
rmse<- RMSE(validation$rating, rating_predicted_reg)
rmses_results<- bind_rows(rmses_results,
                          data.frame(model="Mean and regularized movie, user, and genre bias", RMSE=rmse

rmses_results%>% kable()

# Matrix factorization ----
# Matrix factorization using recosystem
# Convert edx and validation sets to recosystem input format
edx_reco <-  with(edx, data_memory(user_index = userId,
                                   item_index = movieId,
                                   rating     = rating))
validation_reco  <-  with(validation,  data_memory(user_index = userId,
                                                   item_index = movieId,
                                                   rating     = rating))
# construct recommender system object
r <- Reco()

# set parameters
opts_tune<- r$tune(edx_reco)$min

# tuning model parameters
r$train(edx_reco, opts = opts_tune)

# predict ratings based on validation data
rating_predicted<- r$predict(validation_reco, out_memory())

rmse<- RMSE(validation$rating, rating_predicted)
rmses_results<- bind_rows(rmses_results,
                          data.frame(model= "Matrix Factorization",
                                     RMSE= rmse)
)
rmses_results %>% kable
```