HarvardX: PH125.9x Data Science

MovieLens Rating Prediction Project

AANSH SARDANA

30 December, 2020

1 Introduction. -

Recommendation system has been widely applied to e-commerce and personalized recommending services today, such as recommended friends on Facebook, video recommending on Youtube and music recommendations on Itunes and so on. The benefits that a well-designed recommender system could contribute to business is significant. The predicted rate could help provide essentially strong evidence to improve the performance of the entire recommending decisions. In our project, we explored several popular rate-prediction models in recommender system and evaluted and compared which achieved highest possible recommendation accuracy.
This project aims to create a recommendation system by applying a machine learning algorithm that predict movie ratings in validation set. RMSE (the residual mean squared error)is used to evaluate how close predictions of the final model are to the true values.

The project is aimed to create a recommendation model with RMSE < 0.86490

What I have done

The value used to evaluate algorithm performance is the Root Mean Square Error, or RMSE. RMSE is one of the most used measure of the differences between values predicted by a model and the values observed. RMSE is a measure of accuracy, to compare forecasting errors of different models for a particular dataset, a lower RMSE is better than a higher one.
I have made four models that will be developed and be compared using their resulting RMSE in order to assess their quality. The evaluation criteria for this algorithm is a RMSE expected to be lower than 0.8649.

Finally, the best resulting model will be used to predict the movie ratings.

Data setup

First, let's install needed packages and download MovieLens 10M dataset from the grouplens.org.

• [MovieLens 10M dataset - zip file] http://files.grouplens.org/datasets/movielens/ml-10m.zip

```
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")
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_set <- read.table(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_set, movies, by = "movieId")
```

MovieLens dataset will be splitted into 2 subsets that will be the “edx”, a training subset to train the algorithm, and “validation” a subset to test the movie ratings.

```
# The Validation subset will be 10% of the MovieLens data.
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 subset:
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_set, movies, test_index, temp, movielens, removed)
```

2 Analysis and Methods

Data Analysis

To get familiar with the dataset, we analyse the data.
The subset contain the six variables “userID”, “movieID”, “rating”, “timestamp”, “title”, and “genres”.
Each row represent a rating of movie by a user.

#exploration

```
str(edx)
```

Head

```
head(edx) %>%
  print.data.frame()
```

A summary of the subset -

```
summary(edx)
```

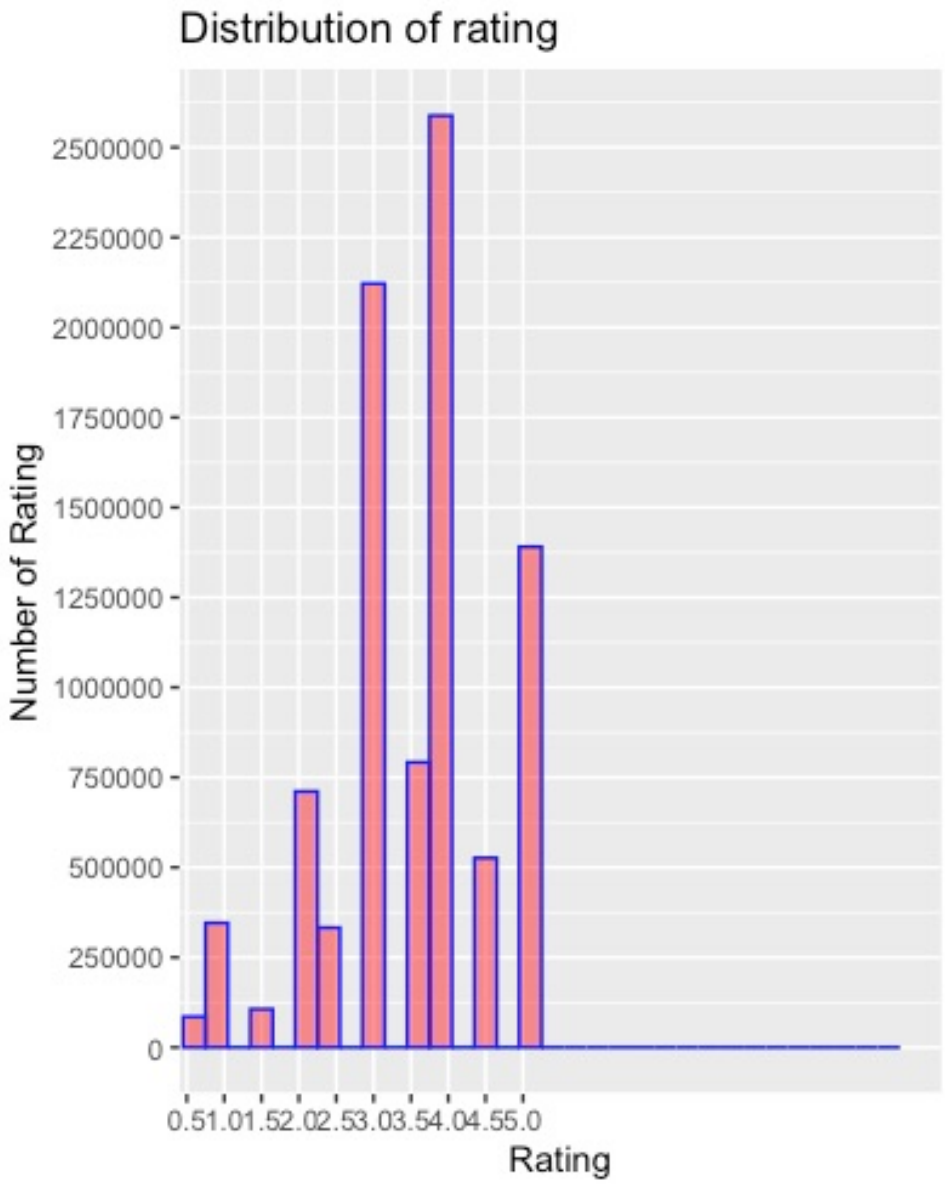
The total of unique users and movies in the subset is about 70.000 and 10.700 respectively:

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

```
> str(edx)
'data.frame':   9000061 obs. of  6 variables:
 $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
 $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
 $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
 $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 838984885 838983707 ...
 $ title    : chr  NA NA NA NA ...
 $ genres   : chr  NA NA NA NA ...
> # Head
> head(edx) %>% print.data.frame()
  userId movieId rating timestamp title genres
1      1     122      5 838985046  <NA>   <NA>
2      1     185      5 838983525  <NA>   <NA>
3      1     231      5 838983392  <NA>   <NA>
4      1     292      5 838983421  <NA>   <NA>
5      1     316      5 838983392  <NA>   <NA>
6      1     329      5 838983392  <NA>   <NA>
> # Summary
> summary(edx)
     userId         movieId          rating       timestamp           title              genres
 Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08   Length:9000061     Length:9000061
 1st Qu.:18122   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08   Class :character   Class :character
 Median :35743   Median : 1834   Median :4.000   Median :1.035e+09   Mode  :character   Mode  :character
 Mean   :35869   Mean   : 4120   Mean   :3.512   Mean   :1.033e+09
 3rd Qu.:53602   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09
 Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
> # Number of unique movies and users in the edx dataset
> edx %>%summarize(n_users = n_distinct(userId),  n_movies = n_distinct(movieId))
  n_users n_movies
1   69878    10677
>
```

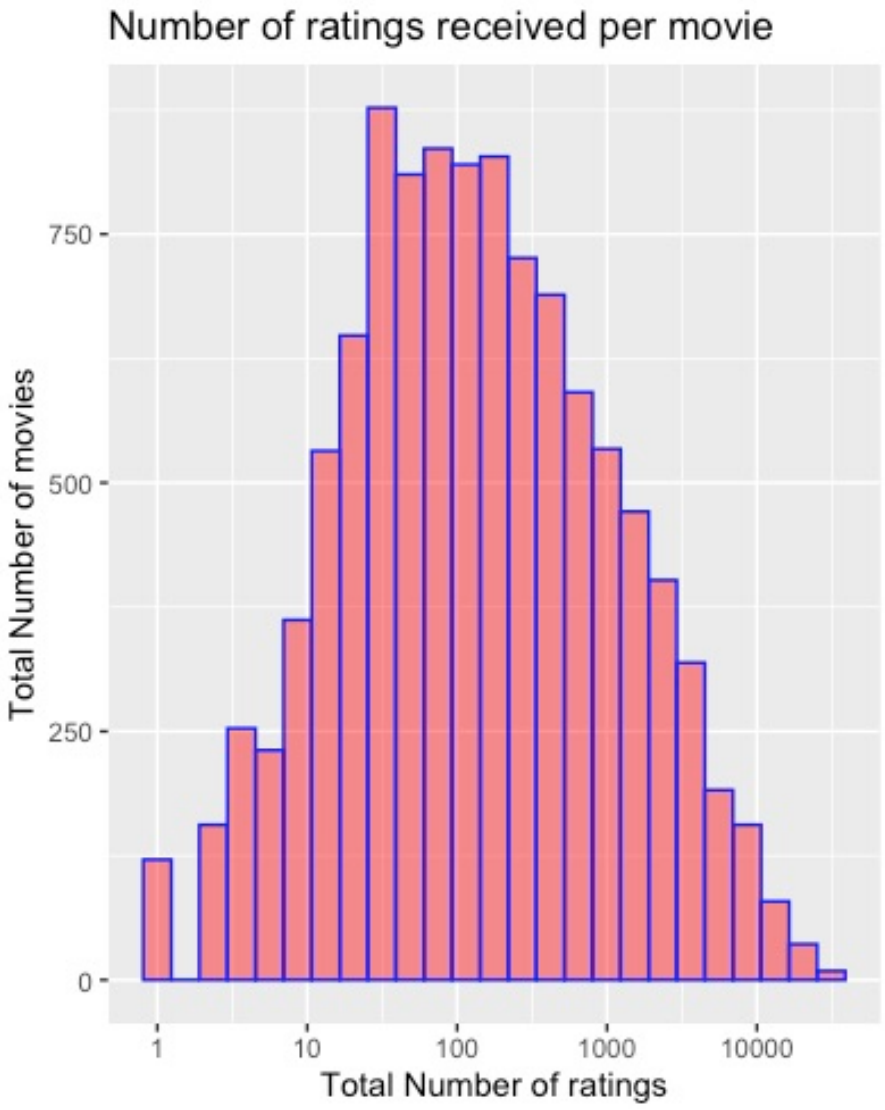
Users rate a movie higher rather than lower as shown by the distribution of ratings below.
Sequence is 4 > 3 > 5 ..... >0.5

```
edx %>%
  ggplot(aes(rating)) +
  geom_histogram(binwidth = 0.30,fill="#fc0303", color = "blue", alpha=0.5) +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  scale_y_continuous(breaks = c(seq(0, 3000000, 250000))) +
  labs(title = "Distribution of rating", y = "Number of Rating", x = “Rating")
```



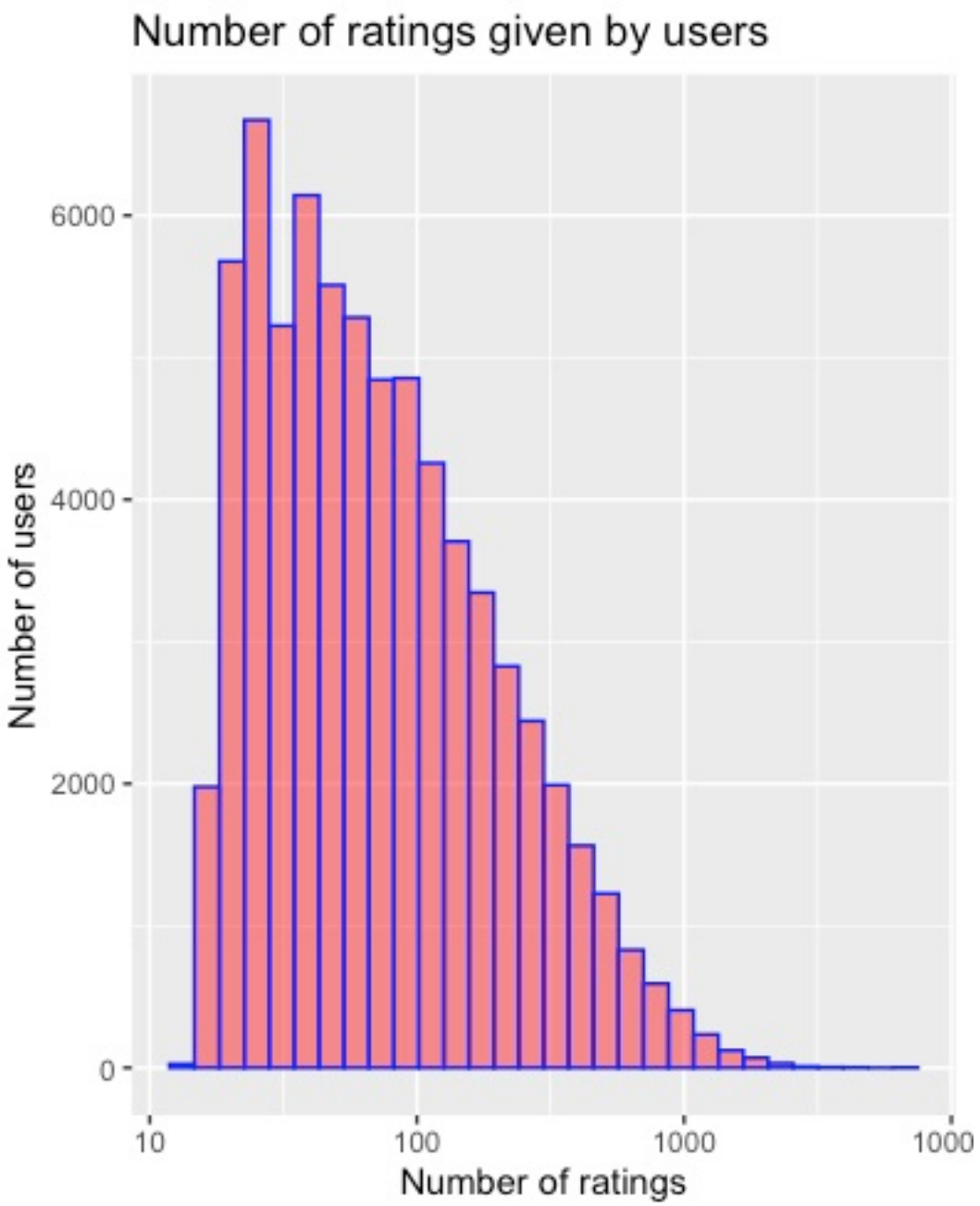
It is also observed that the number of rating also varies from movie to movie .
Some movies are rated much higher whereas some have few ratings while some movies have 1 rating only. These movies with 1 rating are very important for our project study as very low rating numbers might result in wrong estimation for our predictions.
In our database it has been found that 125 movies have been rated once only.
Hence, to rectify this problem , regularisation and a penalty term will be applied in the models in this project.
Regularisations are the techniques used to reduce the error by fitting a function appropriately on the given training set and avoid overfitting.
It is used for tuning the function by adding an additional penalty term in the error function to control excessive fluctuating function such that the coefficients do not take extreme values.

```
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 25, fill="#fc0303", color = "blue", alpha=0.5) +
  scale_x_log10() +
  labs(title = "Number of ratings received per movie", y = "Total Number of movies",
x = "Total Number of ratings")
```



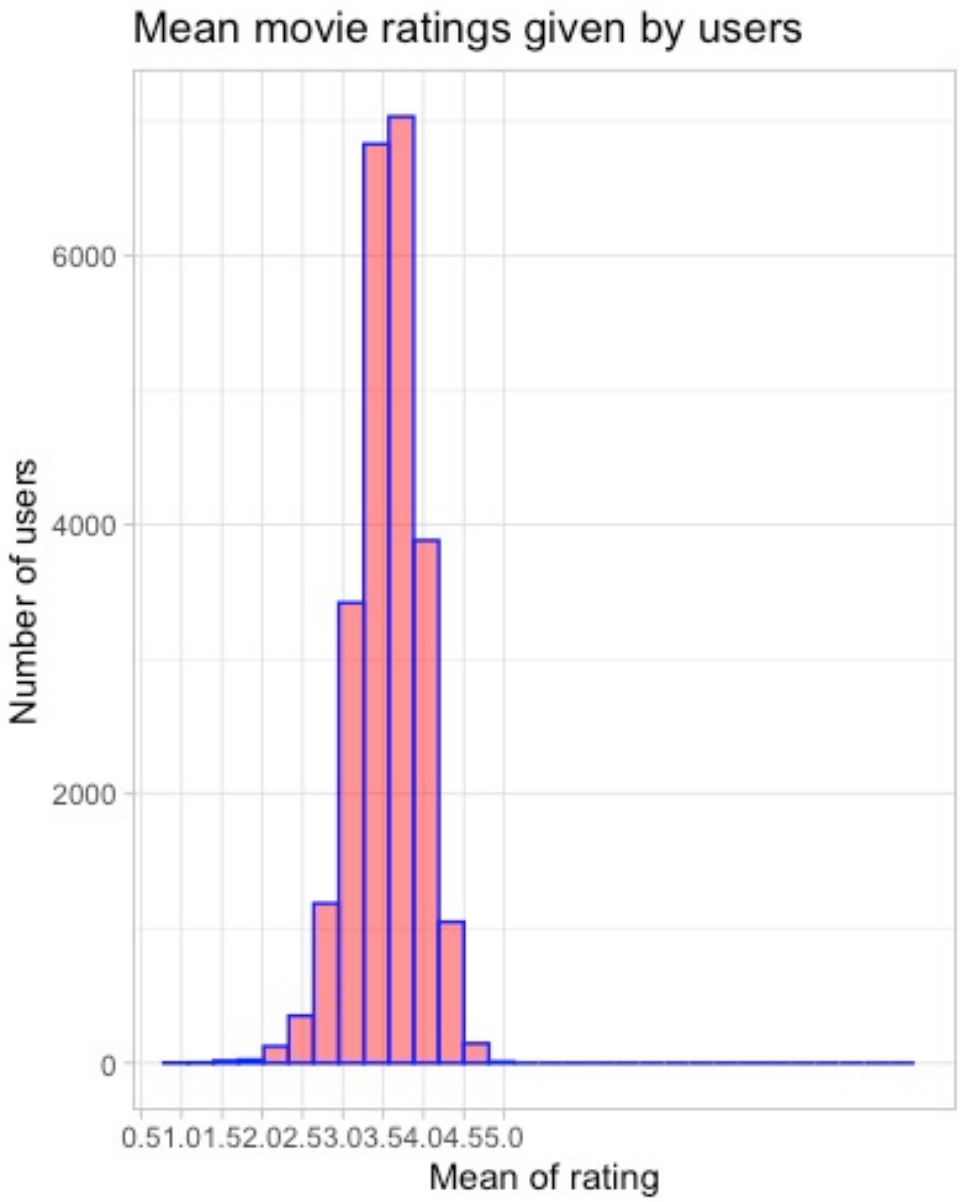
We also need to include user penalty term in our model as most users have rated between 30 and 100

```
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, fill="#fc0303", color = "blue", alpha=0.5) +
  scale_x_log10() +
  labs(title = "Number of ratings given by users", y = "Number of users", x = "Number
of ratings")
```



The visualization below includes only users that have rated atleast 100 movies because some users tend to give much lower ratings and some users tend to give much higher rating than average.

```
edx %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, fill="#fc0303", color = "blue", alpha=0.5) +
  labs(title = "Mean movie ratings given by users", y = "Number of users", x = "Mean
of rating") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  theme_light()
```



3 Modelling Approach

We can concluded that the following parameters have an impact on the rating predictions:
1) movie bias
2) user bias
3) obscure ratings
4) movie genres
5) rating time stamp

I. Average movie rating model

let's predict the same rating for all movies regardless of user which is the average of all ratings.

```
# Computing the dataset's mean rating
me_an <- mean(edx$rating)
# Displaying mean
me_an
```

```
[1] 3.512464
```

we obtain the first naive RMSE:

```
# Testing results based on simple prediction
save_rmse <- RMSE(validation$rating, me_an)
save_rmse
```

```
[1] 1.060651
```

Here, we represent results table with the first RMSE:

```
results_rmse <- data_frame(method = "Average movie rating model", RMSE =
save_rmse)
results_rmse %>% knitr::kable()
```

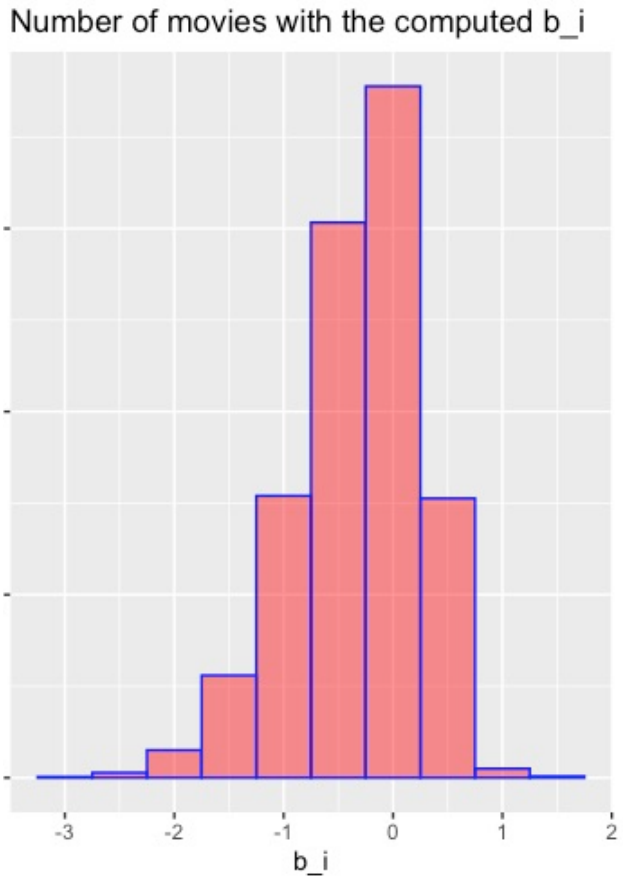
method	RMSE
Average movie rating model	1.060651

This give us our baseline RMSE to compare with next modelling approaches.

II. Movie effect model

Secondly, let's add movie effects to our model as some movies are rated higher then others.
We can use least squares to estimate the movie effect (b_i), but instead we estimate b_i by the average of difference between predicted rating and average rating for each movie .

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - me_an))
movie_avgs
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = .,
fill=I("#fc0303"), color = I("blue"), alpha=I(0.5), ylab = "Number of movies", main =
"Number of movies with the computed b_i")
```



movieId <dbl>	b_i <dbl>
1	0.4137720736
2	-0.3097486590
3	-0.3624568819
4	-0.6321599812
5	-0.4337422252
6	0.2990418828
7	-0.1472776656
8	-0.3953246369
9	-0.5168499360
10	-0.0878410203

1–10 of 10.677 rows

This is called the penalty term movie effect.
Our prediction improve once we predict using this model.

```
predicted_ratings <- me_an +  validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)
predicted_ratings
rmse_model_1 <- RMSE(predicted_ratings, validation$rating)
results_rmse <- bind_rows(results_rmse,
                          data_frame(method="Movie effect model",
                                     RMSE = rmse_model_1 ))
results_rmse
# Checking results
results_rmse %>% knitr::kable()
```

So we have predicted movie rating based on the fact that movies are rated differently by adding the computed b_{i} to $\me_an$.
If an individual movie is on average rated worse that the average rating of all movies me_an , we predict that it will rated lower that me_an by b_i, the difference of the individual movie average from the total average.
We can see an improvement but this model does not consider the individual user rating effect.

method	RMSE
Average movie rating model	1.0606506
Movie effect model	0.9437046

III. Movie and user effect model

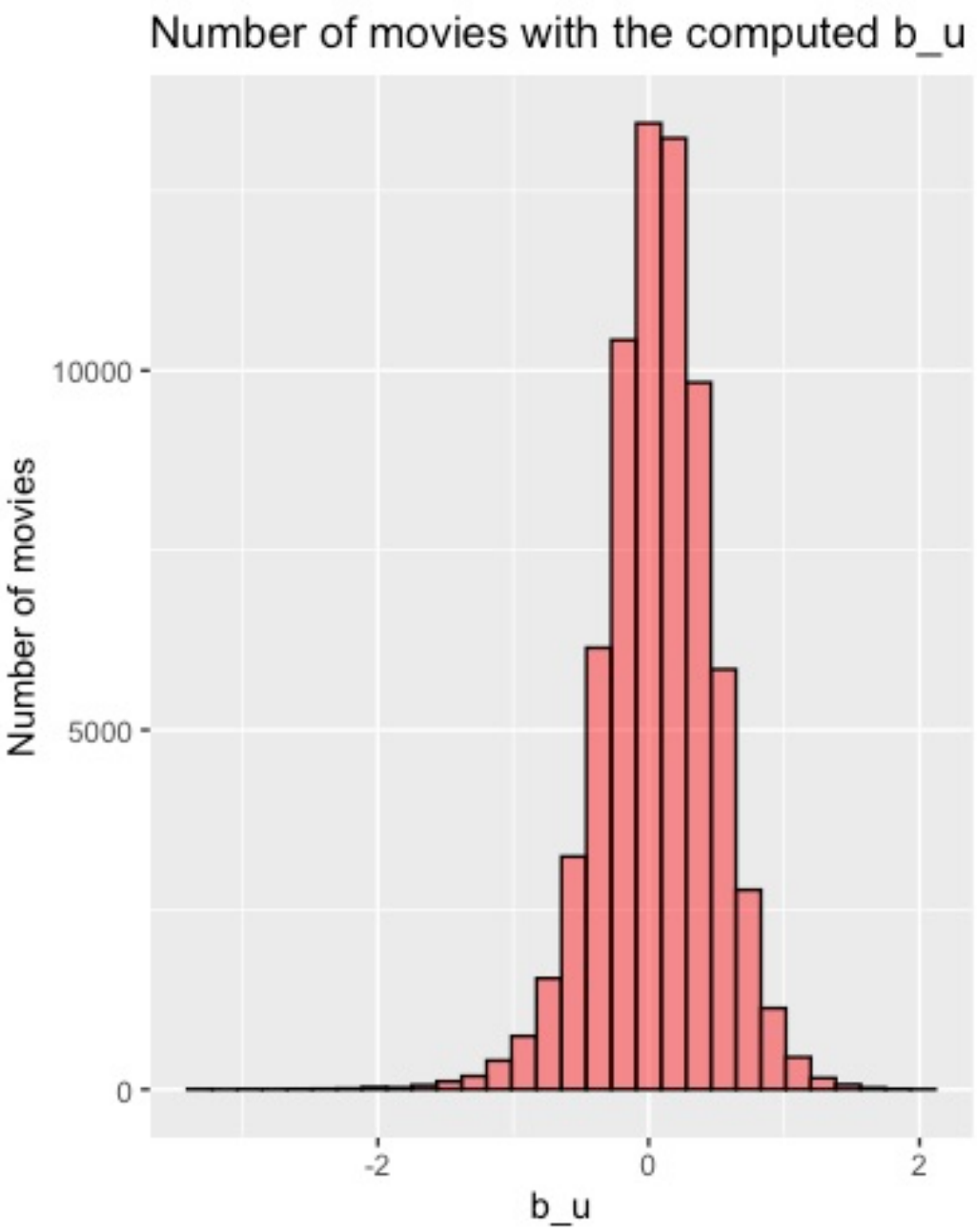
We compute the average rating for user me_an, for those that have rated over 100 movies, said penalty term user effect.

We compute an approximation by computing me_an and b_i, and estimating b_u , as the average of y_u_i - me_an - b_i

```
user_avgs<- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
```

```
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating - me_an - b_i))
user_avgs%>% qplot(b_u, geom ="histogram", bins = 30, data = ., color = I("black"),
fill=I("#fc0303"),alpha=I(0.5),ylab = "Number of movies",main = "Number of
movies with the computed b_u")


user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - me_an - b_i))
```



We can now construct predictors and see RMSE improves:

```
predicted_ratings <- validation%>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = me_an + b_i + b_u) %>%
  pull(pred)

model_2_rmse <- RMSE(predicted_ratings, validation$rating)
results_rmse <- bind_rows(results_rmse,
                          data_frame(method="Movie and user effect model",
                                     RMSE = model_2_rmse))
# Checking result
results_rmse %>% knitr::kable()
```

method	RMSE
Average movie rating model	1.0606506
Movie effect model	0.9437046
Movie and user effect model	0.8655329

In our previous model the RMSE has reduced but there is a major setback in our previous model as we are using only movies for prediciton. The movies rated best and worst are by very few user and in most of cases just by one user. Hence there is alot of uncertainity and the errors will give unexpected and wrong predictions due to high RSME.

IV. Regularized movie and user effect model

In our previous models , we have used standard error method for predicting which has higher degree of uncertainity in results. Therefore in this model we will use regularisation technique and a penalty terms to reduce the effect of overfitting.

Therefore the estimates of b_i and b_u are caused by movies with very few ratings and in some users that only rated a very small number of movies. This can strongly influence the prediction. Now , We should find the value of lamb (which is a turning parameter) to minimise the RMSE .This shrinks the b_i and b_u in case of small number of ratings.

```
lamb <- seq(0, 10, 0.25)
```

```
lamb

# For each lamb,finding b_i & b_u, followed by rating prediction & testing
rmses <- sapply(lamb, function(l){

  me_an <- mean(edx$rating)

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

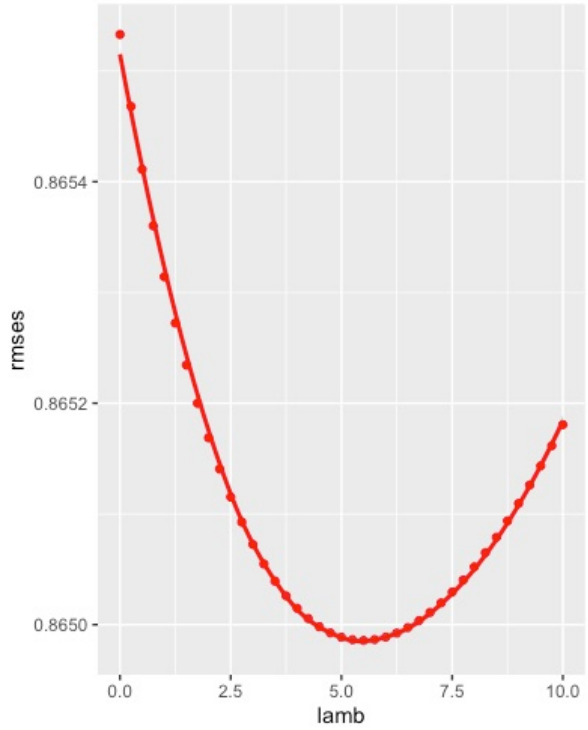
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - me_an)/(n()+l))

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

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

Plotting RMSE vs lambs for selecting the optimal lamb

```
qplot(lamb, rmses, color = I("#fc0303"), geom=c("point", "smooth"))
```



For the full model, the optimal lamb is:

```
lamb<- lamb[which.min(rmses)]
lamb
```

```
  lamb<- lamb[which.min(rmses)]
lamb

```

```
 [1] 5.5
```

For the full model, the optimal lamb is: 5.5

```
results_rmse <- bind_rows(results_rmse, data_frame(method="Regularized movie and user effect model",RMSE = min(rmses)))
```

4 Results

The RMSE values of all the represented models are the following:

```
rmse_results %>% knitr::kable()
```

We therefore found the lowest value of RMSE that is 0.8648170.

```
```{r }

results_rmse <- bind_rows(results_rmse, data_frame(method="Regularized movie and user effect model",RMSE = min(rmses)))

results_rmse
```
```

method <chr>	RMSE <dbl>
Average movie rating model	1.0606506
Movie effect model	0.9437046
Movie and user effect model	0.8655329
Regularized movie and user effect model	0.8648170

5 Conclusion

A machine learning algorithm was built in order to predict movie ratings with MovieLens 10M dataset. The regularized effects of unique users and movies were applied to the model.

The final RMSE result for the recommendation model is 0.8648170. which is better then the target RMSE of 0.86490.