MovieLens Final Capstone Project

Sung Wook Choi

23/04/2020

Introduction

One of the most helpful applcations of machine learning in modern days is the recommendation system, where movies, videos and other media contents are recommended to the users based on their preferences, ratings… etc. Throughout this capstone project an attempt was made to build a model which successfully predicts ratings based off information such as movieids, userids genres, and the years the movies came out.

Methods/Anaylsis

Comments and descriptions are written to explain the procedures, techniques, analysis and other details. This project includes the following procedures/methods: edx/Validation Set Creation, Data Wrangling and Cleaning, Data Analysis and Exploration, Model Building and Final test on the validation set.

1. edx/Validation Set creation

```
################################
# 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")
```

```
## Loading required package: tidyverse
```

```
## ── Attaching packages ───────────────────────────────────────────────────────────────
──────────────────── tidyverse 1.3.0 ──
```

```
## ✓ ggplot2 3.2.1     ✓ purrr   0.3.3
## ✓ tibble  2.1.3     ✓ dplyr   0.8.3
## ✓ tidyr   1.0.0     ✓ stringr 1.4.0
## ✓ readr   1.3.1     ✓ forcats 0.4.0
```

```
## ── Conflicts ────────────────────────────────────────────────────────────────────────
────────────── tidyverse_conflicts() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-projec
t.org")
```

```
## Loading required package: data.table
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
## The following object is masked from 'package:purrr':
##
##     transpose
```

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

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

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"
))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
 colnames(movies) <- c("movieId", "title", "genres")
 movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieI
d],
                                            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")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
# 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)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title",
## "genres")
```

```
 edx <- rbind(edx, removed)

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

2. Data Wrangling/Data Cleaning

```
library(tidyverse)
library(caret)
library(data.table)
if(!require(lubridate)) install.packages("lubridate",
                                         repos = "http://cran.us.r-project.org")
```

```
## Loading required package: lubridate
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday,
##     week, yday, year
```

```
## The following object is masked from 'package:base':
##
##     date
```

```r
library(lubridate)

head(edx)
```

	userId <int>	movieId <dbl>	rating <dbl>	timestamp <int>	title <chr>	genres <chr>
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

6 rows

```r
nrow(edx)
```

```
## [1] 9000055
```

```r
ncol(edx)
```

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

```r
names(edx)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

```r
summary(edx) # Gives general statistical summary of the data
```

```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

```
# We will omit any rows with na as inputs

na.omit(edx)
```

	userId <int>	movieId <dbl>	rating <dbl>	timestamp <int>
1	1	122	5.0	838985046
2	1	185	5.0	838983525
4	1	292	5.0	838983421
5	1	316	5.0	838983392
6	1	329	5.0	838983392
7	1	355	5.0	838984474
8	1	356	5.0	838983653
9	1	362	5.0	838984885
10	1	364	5.0	838983707
11	1	370	5.0	838984596

1-10 of 10,000 rows | 1-5 of 7 columns Previous **1** 2 3 4 5 6 ... 1000 Next

```
na.omit(validation)
```

	userId <int>	movieId <dbl>	rating <dbl>	timestamp <int>	title <chr>
1	1	231	5.0	838983392	Dumb & Dumber (1994)
2	1	480	5.0	838983653	Jurassic Park (1993)
3	1	586	5.0	838984068	Home Alone (1990)
4	2	151	3.0	868246450	Rob Roy (1995)
5	2	858	2.0	868245645	Godfather, The (1972)
6	2	1544	3.0	868245920	Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
7	3	590	3.5	1136075494	Dances with Wolves (1990)
8	3	4995	4.5	1133571200	Beautiful Mind, A (2001)
9	4	34	5.0	844416936	Babe (1995)
10	4	432	3.0	844417070	City Slickers II: The Legend of Curly's Gold (1994)

1-10 of 10,000 rows | 1-6 of 7 columns Previous **1** 2 3 4 5 6 ... 1000 Next

```r
# Let's clean the data a little bit. We will change the column timestamp to dates using
 the lubridate package, to see the exact date when the movies were rated

library(lubridate)
edx <- edx %>% mutate(dates = as_datetime(timestamp)) %>% select(-timestamp)
validation <- validation %>% mutate(date = as_datetime(timestamp)) %>% select(-timestam
p)
head(validation)
```

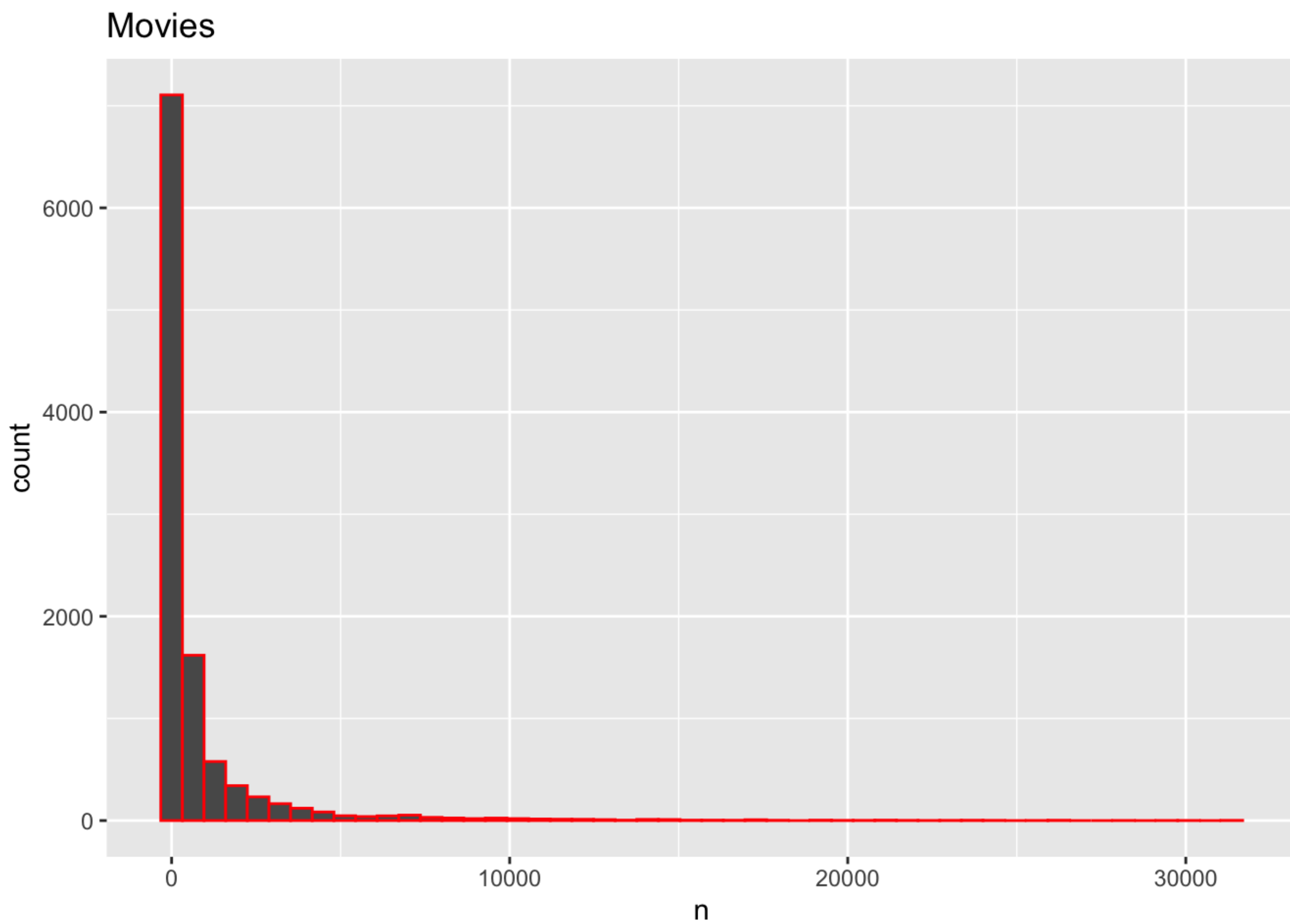
	userId <int>	movieId <dbl>	rating <dbl>	title <chr>
1	1	231	5	Dumb & Dumber (1994)
2	1	480	5	Jurassic Park (1993)
3	1	586	5	Home Alone (1990)
4	2	151	3	Rob Roy (1995)
5	2	858	2	Godfather, The (1972)
6	2	1544	3	Lost World: Jurassic Park, The (Jurassic Park 2) (1997)

6 rows | 1-5 of 7 columns

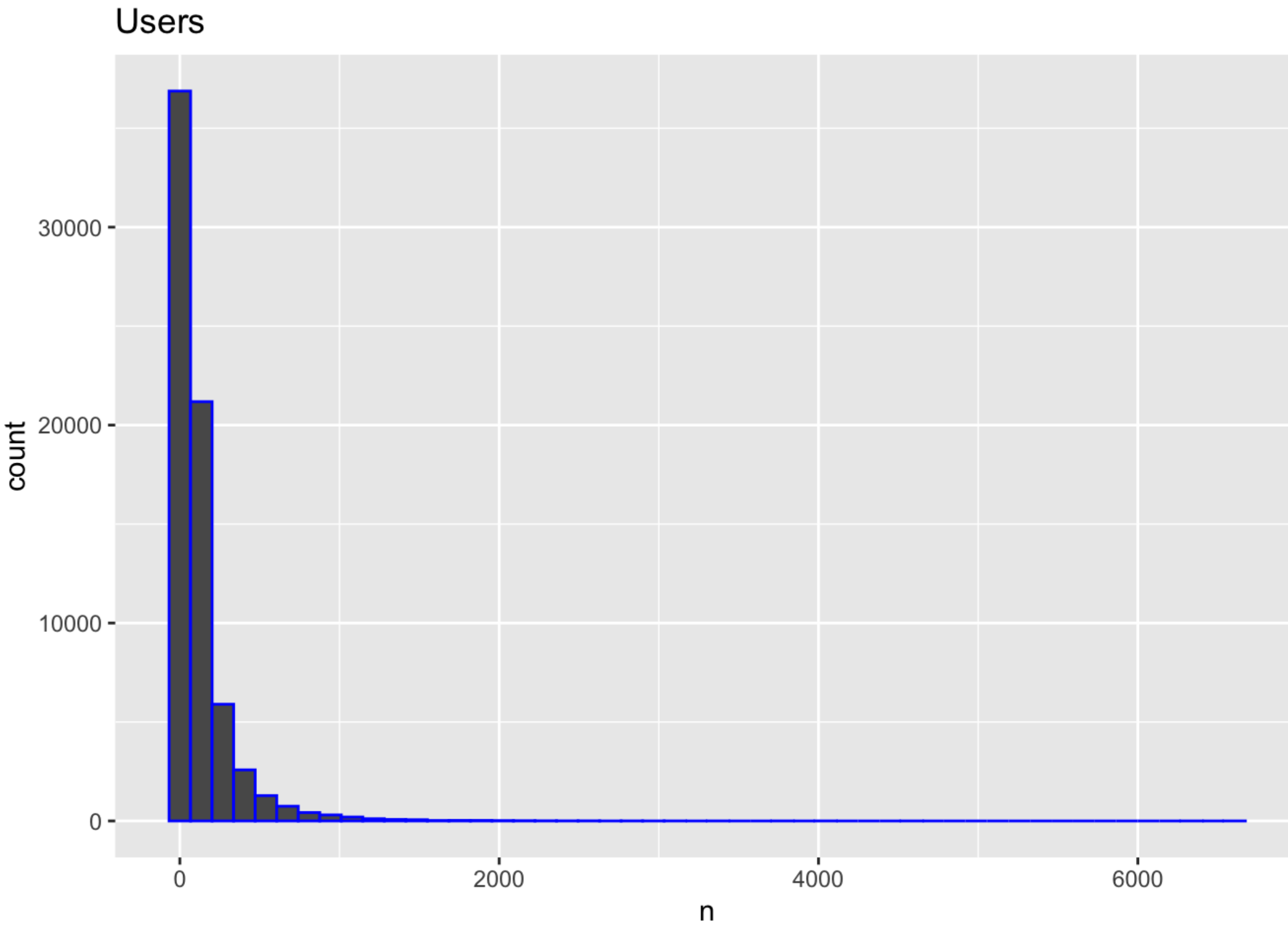
```r
# Create a new column, years
edx  <- edx %>% mutate(years = as.numeric(str_sub(title,-5,-2)))
# extract years the movie came out and create a new column called years

results <- tibble()
#create a tibble which will organize all the RMSE's for different models
```

3. Data Analysis/Exploration

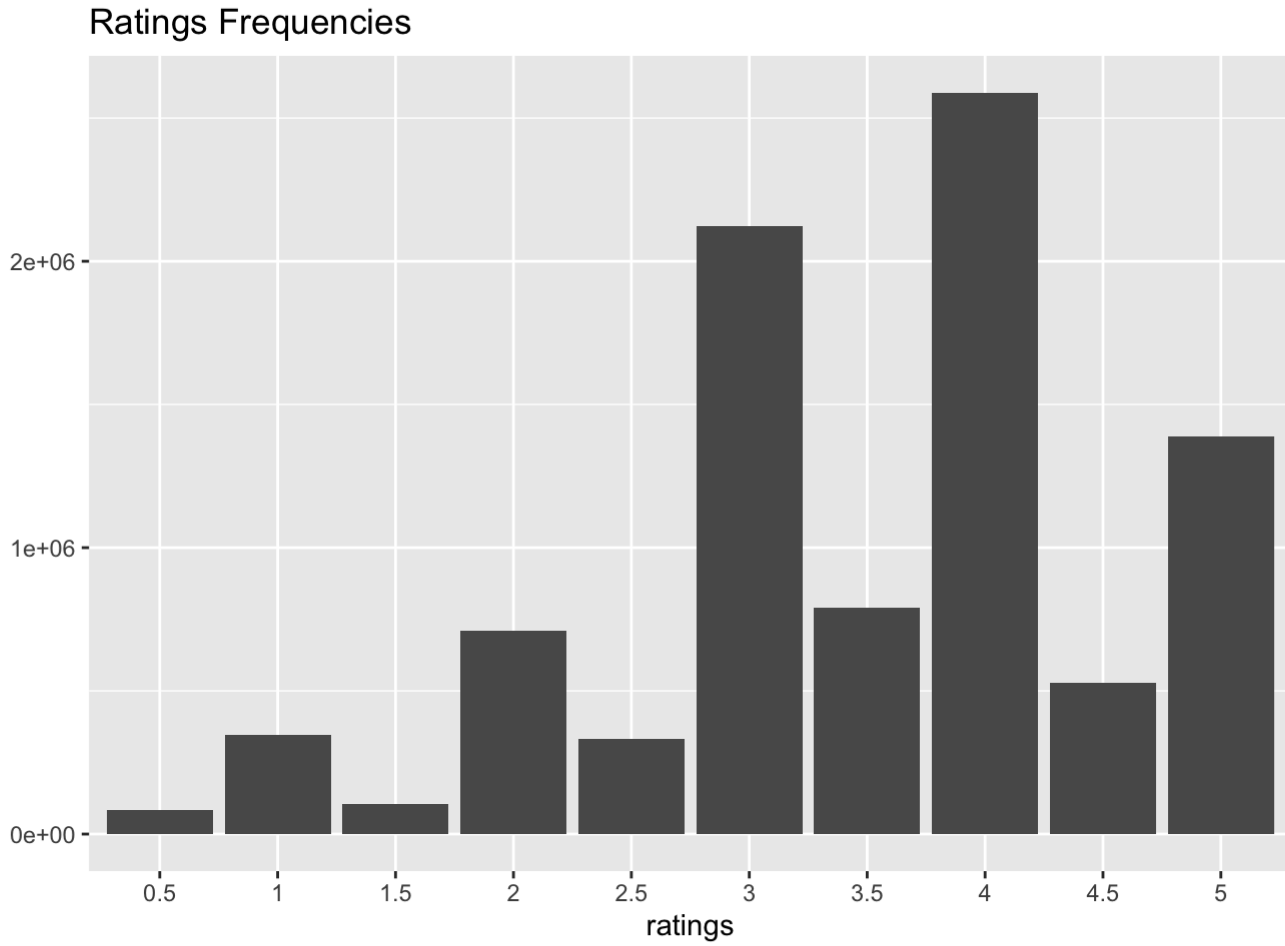
```r
if(!require(tidyverse)) install.packages("ggplot2", repos = "http://cran.us.r-project.or
g")
library(ggplot2)

# Let's take a closer look at the edx data and see if there is any bias within the data
edx %>%
     dplyr::count(movieId) %>%
     ggplot(aes(n)) +
     geom_histogram(bins = 50, color = "red") +
     ggtitle("Movies")
```




```
# We can see that some movies are rated more than others

edx %>%
     dplyr::count(userId) %>%
     ggplot(aes(n)) +
     geom_histogram(bins = 50, color = "blue") +
     ggtitle("Users")
```




```
# Some users rate the movies more often than others
# Let's also take a look at the frequency/distribution of the ratings

ratings <- as.vector(edx$rating)
ratings <- factor(ratings)
qplot(ratings) +
  ggtitle("Ratings Frequencies")
```




```
# There is a high tendency that people often give out 3,4 as ratings
```

```
# We can also analyze the relationship between the variable time and other predictors
# We can take a look at the relationship between years the movies were released and the
 mean ratings of each year

edx %>% group_by(years) %>%
   summarize(mean_rating = mean(rating)) %>%
   ggplot(aes(years, mean_rating)) +
   geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```


```
# Used Loess method to smooth by default
# We can see a generally see the that the average ratings for more recent movies are low
er and movies that were in the mid-early 1900's have higher mean ratings
```

4. Model Building

```
# Create a functon called 'RMSE'
RMSE <- function(actual, predicted){
  sqrt(mean((actual-predicted)^2))
}

# Clean the data for the validation set
validation <- validation %>% mutate(years = as.numeric(str_sub(title,-5,-2)))

# Splitting edx into test and training set
set.seed(1996)
test_index2 <- createDataPartition(edx$rating, times = 1, p = 0.1, list = FALSE)
temporary_test <- edx %>% slice(test_index2)
train <- edx %>% slice(-test_index2)

# Making sure testset and train set have same movieIds and userIds
test <- temporary_test %>% semi_join(train, by = "movieId") %>%
   semi_join(train, by = "userId")

# Putting the removed rows back into the training set
removed <- anti_join(temporary_test, test)
```

```
## Joining, by = c("userId", "movieId", "rating", "title", "genres", "dates",
## "years")
```

```
train <- rbind(train, removed)

test_ratings <- test$rating # ratings in test set
```

An extremely Simple model where ratings are pulled out randomly. We have a vector of ratings all the way from 0 to 5.0. These will be sampled randomly to predict the rating of a movie.

```
set.seed(2020)
random <- sample(c(0, 0.5, 1, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0), length(test_ratin
gs), replace = TRUE)
RMSE(test_ratings,random)
```

```
## [1] 2.156021
```

```
# RMSE turns out 2.156021 which is terrible considering that the prediction could differ
by up to two stars!
# Will test it on the validation set

random <- sample(c(0, 0.5, 1, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0), length(validation
$rating), replace = TRUE)
rmse_random <- RMSE(validation$rating, random)
results <- bind_rows(results, data_frame(method="Random Model",  RMSE = rmse_random))
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

Simple model using just the yearly average of the entire ratings in the edx dataset. First we use a model that looks like the following (u, i subscripts for users and movies respectively):

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

In this model we assume that they all have same ratings for all movies and users, where they differ just by random errors. Epsilon represents independent random errors.

```
mu <- mean(train$rating)
RMSE(test_ratings, mu)
```

```
## [1] 1.059691
```

```
# RMSE has now decreased to 1.59691 which is much better, but more improvement could be
 made to this model
# Let's test it on our validation set to obtain our RMSE

mean_rmse <- RMSE(validation$rating, mu)
results <- bind_rows(results, data_frame(method= "Mean Model",  RMSE = mean_rmse))
```

Instead of assuming the same ratings for all movies and users, We will incorporate the term b_i in the term which is the average rating for movie i (movie specific effect):

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

```
permovie_averages1 <- train %>% group_by(movieId) %>%
   summarize(bi = mean(rating - mu))

model1_prediction <- test %>% left_join(permovie_averages1, by='movieId') %>%
   mutate(prediction = mu + bi)

RMSE(test_ratings, model1_prediction$prediction)
```

```
## [1] 0.9430351
```

```
# This was the RMSE for the test set, we will now obtain RMSE for the validation set

model1_prediction_valid <- validation %>% left_join(permovie_averages1, by='movieId') %
>%
   mutate(prediction = mu + bi)
model1_rmse <- RMSE(validation$rating, model1_prediction_valid$prediction)
results <- bind_rows(results, data_frame(method= "Movie Specific Effect Model",  RMSE =
 model1_rmse))
```

We could make an improvement to our model by further incorporating the userId specific effects b_u.

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

```
permovie_averages2 <- train %>%
   left_join(permovie_averages1, by='movieId') %>%
   group_by(userId) %>%
   summarize(bu = mean(rating - mu - bi))

model2_prediction <- test %>% left_join(permovie_averages2, 'userId') %>%
   left_join(permovie_averages1, by='movieId') %>%
   mutate(prediction = mu + bi + bu)

RMSE(test_ratings, model2_prediction$prediction)
```

```
## [1] 0.8651999
```

```
# We did a better job at estimating the rating by incorporating the user specific effect
to our model
# Let's give it a try on our validation set

model2_prediction_valid <- validation %>% left_join(permovie_averages1, by='movieId') %
>%
   left_join(permovie_averages2, by = 'userId') %>%
   mutate(prediction = mu + bi + bu)

model2_rmse <- RMSE(validation$rating, model2_prediction_valid$prediction)
results <- bind_rows(results, data_frame(method= "Movie + userId Specific Effect Model",
RMSE = model2_rmse))
```

We still could do better, we will incorporate the year effect b_y. Now our model looks like the following.

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

```
permovie_averages5 <- train %>%
   left_join(permovie_averages1, by='movieId') %>%
   left_join(permovie_averages2, by = 'userId') %>%
   group_by(years) %>%
   summarize(by = mean(rating - mu - bi - bu))

model5_prediction <- test %>% left_join(permovie_averages2, 'userId') %>%
   left_join(permovie_averages1, 'movieId') %>%
   left_join(permovie_averages5, 'years') %>%
   mutate(prediction = mu + bi + bu + by)

RMSE(test_ratings, model5_prediction$prediction)
```

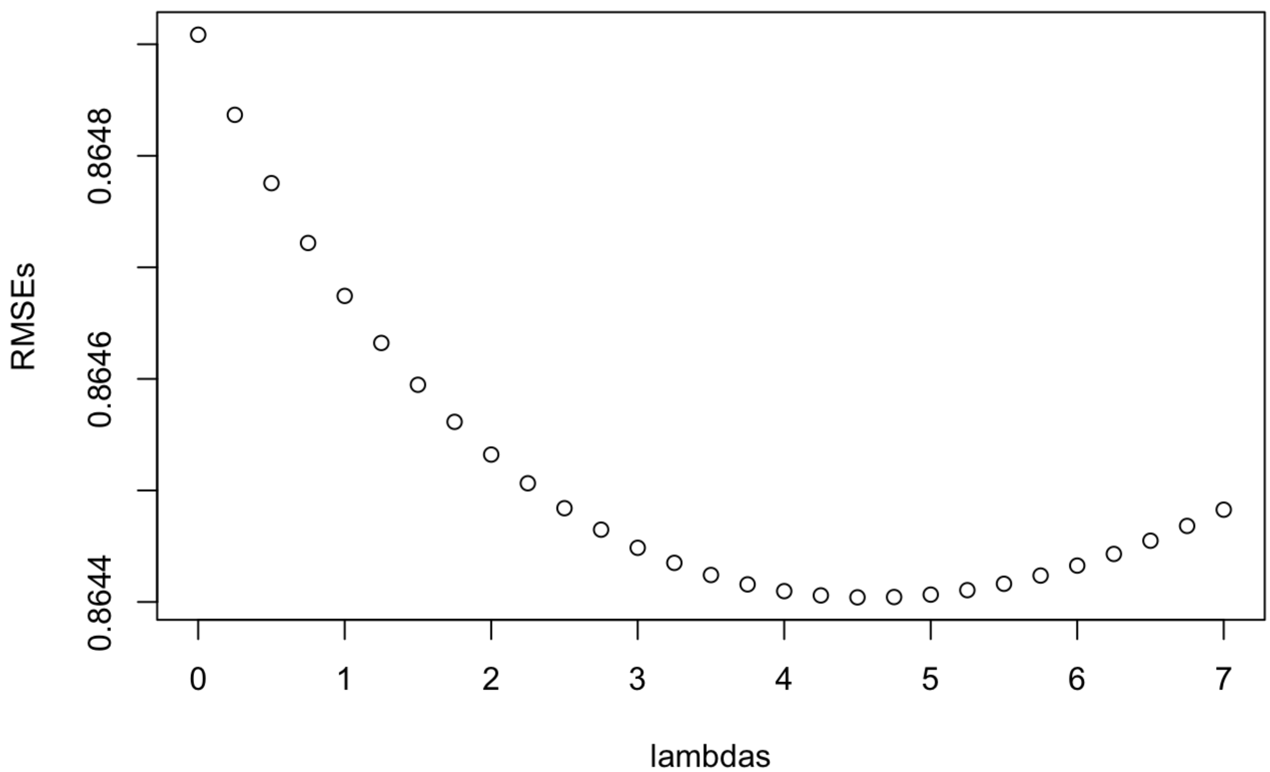
```
## [1] 0.8649086
```

```
# By incorporating the year effect we were able to make an improvement on our model

# Let's show that using our validation set
model5_prediction_valid <- validation %>% left_join(permovie_averages1, by='movieId') %
>%
   left_join(permovie_averages2, by = 'userId') %>%
   left_join(permovie_averages5, 'years') %>%
   mutate(prediction = mu + bi + bu + by)

model5_rmse <- RMSE(validation$rating, model5_prediction_valid$prediction)
results <- bind_rows(results, data_frame(method= "Movie + userId + year Specific Effect
 Model",  RMSE = model5_rmse))
```

Regularization Based Approach: There were biases in our data; some movies were rated more often than others while some users rated movies more often than others. Furthermore, ratings on average were lower for more mordern movies. We will use the regularization based approach in order to minimize these effects on our results or bias. We will use cross-validation method to pick our best lambda which allows us to obtain. We will use regularization on movie, userId, and year specific effects.

```
lambdas <- seq(0,7,0.25)
# Here the lambdas are tuning parameters and we will find the best lambda through cross
 validation method
# For each lambda, bi & bu is calculated and ratings are predicted & tested against the
 testset
# Cross validation code requires some time to run

list_RMSE <- function(lambda){

mu <- mean(train$rating)

permovie_averages3 <- train %>%
   group_by(movieId) %>%
   summarize(bi = sum(rating - mu)/(n() + lambda)) # movie specific effect regularized

permovie_averages4 <- train %>%
   left_join(permovie_averages3, by='movieId') %>%
   group_by(userId) %>%
   summarize(bu = sum(rating - mu - bi)/(n() + lambda)) # userId specific effect reguala
rized

permovie_averages6 <- train %>%
      left_join(permovie_averages3, by='movieId') %>%
      left_join(permovie_averages4, by ='userId') %>%
      group_by(years) %>%
      summarize(by = sum(rating - mu - bi - bu)/(n() + lambda)) # year specific effect r
egualarized
# predict
test_prediction <- test %>% left_join(permovie_averages3, by = 'movieId') %>%
      left_join(permovie_averages4, by ='userId') %>%
      left_join(permovie_averages6, by ='years') %>%
      mutate(prediction = mu + bi + bu + by)

   RMSE(test_ratings, test_prediction$prediction)
}
```

```
RMSEs <- sapply(lambdas, list_RMSE)
plot(lambdas, RMSEs)
```




```
lambdas[which.min(RMSEs)]
```

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

```
# Lambda which minimized RMSE the most (optimal RMSE) against the test data was 4.5
# Now that we have our model lambda, we will test it out on our validation set
```

5. Final Test on Validation Set

With our newly obtained lambda we will test our model on our validation set to obtain our final RMSE.

```
mu <- mean(train$rating)

permovie_averages3 <- train %>%
   group_by(movieId) %>%
   summarize(bi = sum(rating - mu)/(n() + 4.5)) # movie specific effect regularized

permovie_averages4 <- train %>%
   left_join(permovie_averages3, by='movieId') %>%
   group_by(userId) %>%
   summarize(bu = sum(rating - mu - bi)/(n() + 4.5)) # userId specific effect regualariz
ed

permovie_averages6 <- train %>%
      left_join(permovie_averages3, by='movieId') %>%
      left_join(permovie_averages4, by ='userId') %>%
      group_by(years) %>%
      summarize(by = sum(rating - mu - bi - bu)/(n() + 4.5)) # year specific effect regu
alarized

validation_prediction <- validation %>% left_join(permovie_averages3, by = 'movieId') %
>%
      left_join(permovie_averages4, by ='userId') %>%
      left_join(permovie_averages6, by ='years') %>%
      mutate(prediction = mu + bi + bu + by)
```

```
final <- RMSE(validation$rating, validation_prediction$prediction)
results <- bind_rows(results, data_frame(method= "Regularization on Movie + userId Speci
fic Effect Model",  RMSE = final))
```

Results/Model Performance

There was a significant improvement of RMSE through addition of different effects to models. Our very first model which was just a random generation of ratings (certainly not what users want), had absurdly high RMSE of 2.156021. When we assumed that all the ratings were equal to the mean of the entire ratings, we obtained RMSE of 1.059691. Once we added movie specific and user specific effects to our mode our models certainly improved as the RMSE's decreased to 0.9430351 and 0.8651999 respectively (Up to this point we tested the models on our test set). Once we used our regularization method and cross validation method to select the optimal lambda of 4.5. Once we used regularization method on our final validation set, we obtained RMSE just extremely near 0.86490, which tells us that the model works very well and is trustworthy enough to predict ratings of the movies for the users.

```
results %>% knitr::kable() #final results
```

method	RMSE
Random Model	2.1555584
Mean Model	1.0612018
Movie Specific Effect Model	0.9439815
Movie + userId Specific Effect Model	0.8658185

method	RMSE
Movie + userId + year Specific Effect Model	0.8654779
Regularization on Movie + userId Specific Effect Model	0.8649037

Conclusion

Throughout this edx project, I had the opportunity to have a great practice to brush up on skills such as data wrangling, exploration, critical thinking and build models that could predict outcomes. I have attempted to come up with an algorithm, given multiple features that predicts movie ratings. In conclusion model which we obtained from regularized effects on movie, user effect and years the movies were released were the most accurate by far, in terms of its ability to predict movie ratings. I don't see any limitations, as in factors that could have hindered my results. This project could have potentially a positive impact on the users of media platforms such as Netflix or Youtube, in a sense that this algorithm could allow the users to have more positive experience on the platform as more accurate recommendations could be made.