HarvardX Data Science Capstone Project: MovieLens

Sharmin Shabnam

1/10/2020

Contents

1 Introduction

Streaming services such as Netflix frequently amazes us with how they can almost perfectly capture our taste in movies. These services collect data regarding movies watched by a given user and their respective ratings. The rating system is based on a scale from one to five stars: one star suggests the user did not like the movie, whereas five star rating suggests they enjoyed the movie. A Movie recommendation System is built upon these data to predict what rating a specific user would give to a specific movie. Subsequently, movies that are predicted to be given high ratings are suggested to that user.

In this project, a movie recommendation system is built using a stable benchmark dataset of movies rated by millions of users. For the purpose of computational feasibility, a subset(MovieLens 10M Dataset, released 1/2009) of the original dataset (containing 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users) is used.

The goal of this project is to predict ratings for a set of users whose actual ratings are contained in a validaiton set and previously unused by the prediction model. We check the performance of our predictions against RMSE (Root Mean Squared Error) which is given as [1]:

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

Where: $y_{u,i}$ = the rating for movie i by user u $\hat{y}_{u,i}$ = our prediction with recommendation system N = the number of user/movie combinations and the sum occurring over all these combinations.

RMSE can be interpreted as the typical error occured while predicting a movie rating. An RMSE greater than 1 means our typical error is larger than one star, which indicates the model's performance is not within the acceptable range and could be imporved upon.

```
# The RMSE function that will be used in this project is:
RMSE <- function(true_ratings = NULL, predicted_ratings = NULL) {
    sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

2 Data Exploration

2.1 Data set download and variable exploration

The code for dataset import, extraction and variable selection was provided by Edx which is used below to create training and validation set for further model building.

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

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

#Read ratings.dat and movie.dat file and assign column names
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")

#Create dataframe for movies
movies <- as.data.frame(movies) %>%
  mutate(movieId = as.numeric(levels(movieId))[movieId],
         title = as.character(title),
         genres = as.character(genres))
#Join movies and ratings dataframe
movielens <- left_join(ratings, movies, by = "movieId")
```

2.2 Creating partitions of training and test data set

The following code was used to create data partitions.

```
# Validation set will be 10% of MovieLens data
set.seed(1000, sample.kind="Rounding")
```

```
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]


#userId and movieId in validation set are
#implemented according to training set edx
 validation <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Addition of 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)

#Remove unnecessary data and clean up memory
rm(dl, ratings, movies, test_index, temp, movielens, removed)
gc()
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells 11167025 596.4   14486860  773.7  22137590 1182.3
## Vcells 94735407 722.8  318230466 2428.0 285844528 2180.9
```

2.3 Training Set Exploration

The training data is used in the following plots to observe the variability of ratings with regard to movies, users and time.

```
#edx is the training Set and Validation is the test set
#Summarise Data
head(edx, 5)
```

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	231	5	838983392	Dumb & Dumber (1994)	Comedy
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi

```
summary(edx)
```

```
##      userId         movieId          rating       timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.50   Min.   : 789652009
##  1st Qu.:18112   1st Qu.:  648   1st Qu.:3.00   1st Qu.: 946769920
##  Median :35730   Median : 1834   Median :4.00   Median :1035493058
##  Mean   :35866   Mean   : 4120   Mean   :3.51   Mean   :1032615440
##  3rd Qu.:53602   3rd Qu.: 3627   3rd Qu.:4.00   3rd Qu.:1126744764
##  Max.   :71567   Max.   :65133   Max.   :5.00   Max.   :1231131303
##     title              genres
##  Length:9000064     Length:9000064
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
```

```
##
##
```

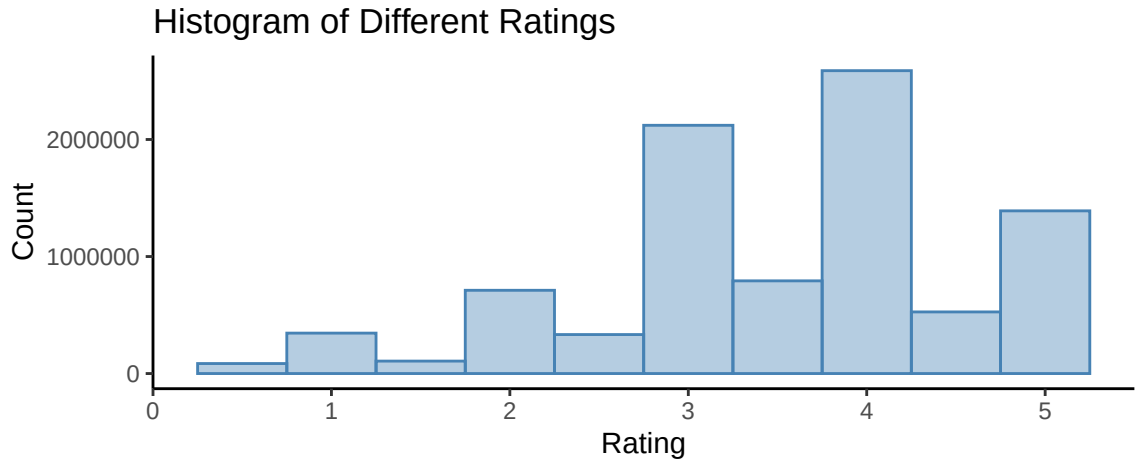
```
#Users, Movies and Genres in Database
edx %>% summarise(
  uniq_users = n_distinct(userId),
  uniq_movies = n_distinct(movieId),
  uniq_genres = n_distinct(genres))
```

uniq_users	uniq_movies	uniq_genres
69878	10677	797

```
#Mean of all the ratings
mean(edx$rating)
```

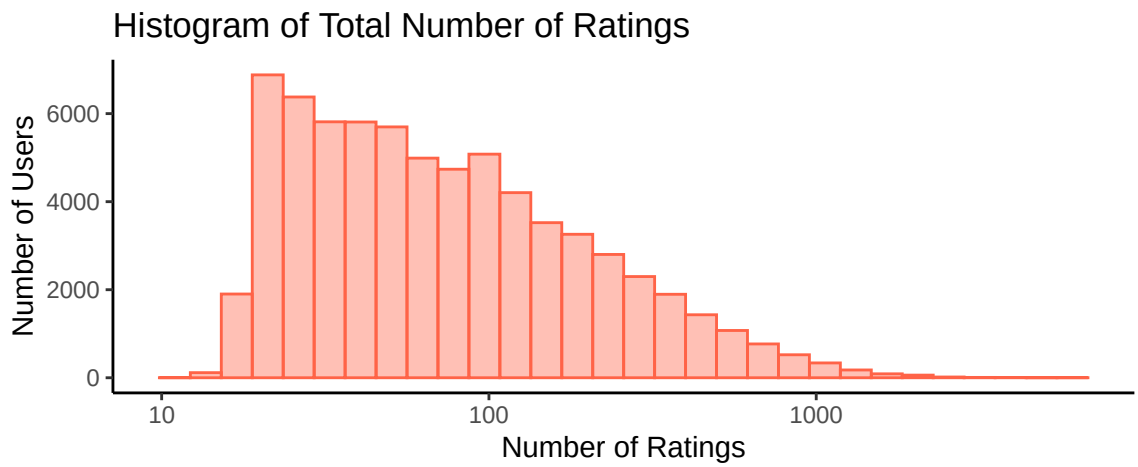
```
## [1] 3.51243
```

```
#Exploratory Data Analysis
#Histogram of Different Ratings
edx %>%
  ggplot(aes(rating)) + theme_classic()+
  geom_histogram(binwidth = 0.5, color = "steelblue",fill="steelblue", alpha=0.4) +
  xlab("Rating") +
  ylab("Count") +
  ggtitle("Histogram of Different Ratings")
```



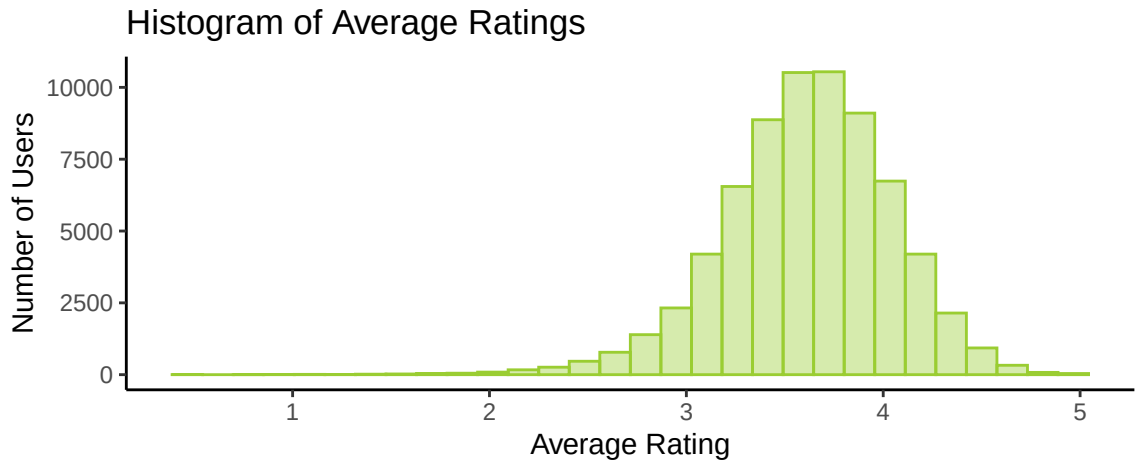

```
#Histogram of Total Number of Ratings
edx %>%
  count(userId) %>%
  ggplot(aes(n)) + theme_classic()+
  geom_histogram(color = "tomato",fill="tomato", alpha=0.4) +
  scale_x_log10() +
  xlab("Number of Ratings") +
  ylab("Number of Users") +
  ggtitle("Histogram of Total Number of Ratings")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



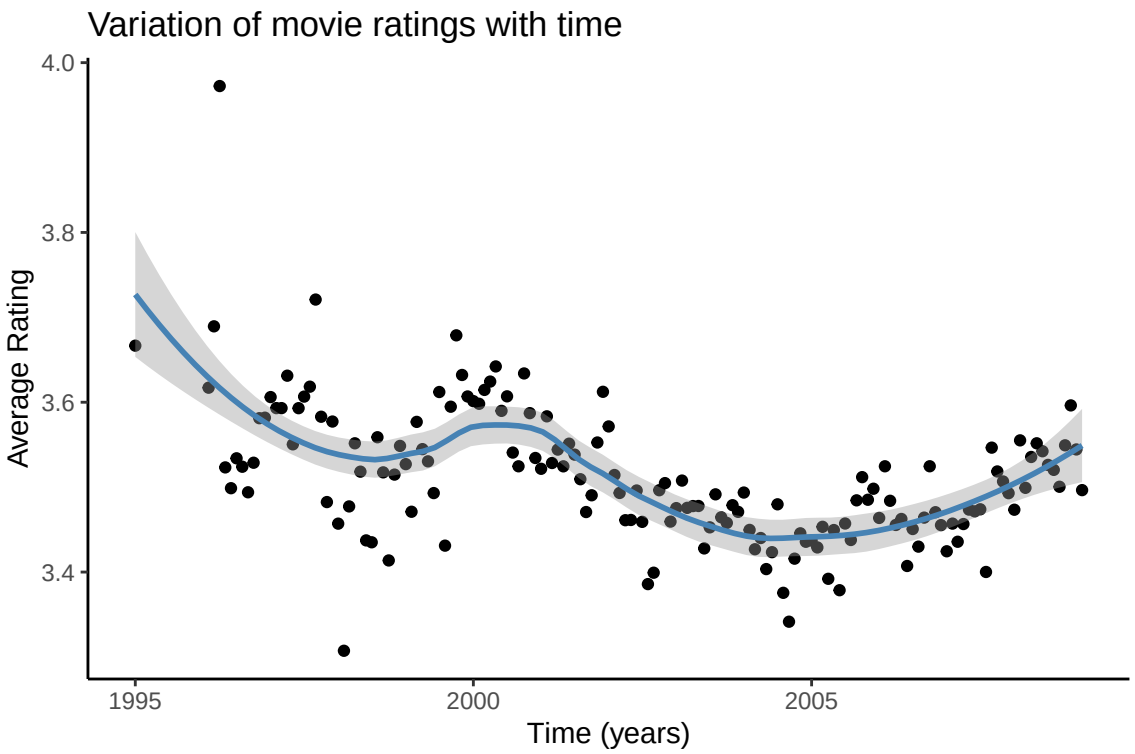

```
#Histogram of Average Ratings
edx %>%
  group_by(userId) %>%
  summarise(mean_r = mean(rating)) %>%
  ggplot(aes(mean_r)) + theme_classic()+
  geom_histogram(color = "yellowgreen",fill="yellowgreen", alpha=0.4) +
  ggtitle("Histogram of Average Ratings") +
  xlab("Average Rating") +
  ylab("Number of Users")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
#Variation of movie ratings with time
edx %>%
  mutate(timestamp = round_date(as_datetime(timestamp), unit = "month")) %>%
  group_by(timestamp) %>% summarize(mean_r = mean(rating)) %>%
  ggplot(aes(timestamp, mean_r)) + theme_classic() +
  geom_point() +
  geom_smooth(color='steelblue', span = 0.5) +
  ggtitle("Variation of movie ratings with time") +
  ylab("Average Rating") +
  xlab("Time (years)")
```

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



3 Methods and analysis

3.1 Regression Model

As mentioned in course texbook [1], a similar approach of Linear Reression Model was first implemented to build prediction algorithm for the simplest possible recommendation systems.

3.1.1 Naive Mean-Baseline Model

To initiate model building for recommendation system, a simple analysis based on the overall mean of the ratings is used in this section. The model named as Naive Mean-Baseline Model, predicts the same rating (=overall mean) for all the users in the database. The prediction is defined as:

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

where: $Y_{u,i}$ = prediction for movie i by user u, μ = is the "true" rating for all movies, $\varepsilon_{u,i}$ = independent errors sampled from the same distribution centered at 0.

```
#Simple Prediction using Mean Rating
mu <- mean(edx$rating)
mu
```

```
## [1] 3.51243
```

```
#Calculate RMSE of Naive Mean-Baseline Model
rmse_naive_mean <- RMSE(validation$rating, mu)
rmse_naive_mean
```

```
## [1] 1.06093
```

```
#Save Results in a Data Frame
rmse_results <- data_frame(Method = "Naive Mean Baseline Model",
                           RMSE = as.numeric(rmse_naive_mean)) %>%
                            mutate_if(is.numeric, round, digits = 3)

#Clean up memory
rm(rmse_naive_mean)
gc()
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells 11288197 602.9   17424232  930.6  22137590 1182.3
## Vcells 94973270 724.6  321889858 2455.9 321889858 2455.9
```

3.1.2 Model with Movie effects

Some movies are very popular and hence rated more than others movies in the database. The model in this section includes a bias term for each movie based on the difference between the movies mean rating and the overall mean rating.

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

where: b_i = average ranking for movie i.

```
#Estimate movie effects, b_i
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - mu))


#Make predictions using Movie Effects
predicted_ratings <- mu + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_i)


#Calculate RMSE of Model with Movie effects
rmse_movie_effects <- RMSE(predicted_ratings, validation$rating)
rmse_movie_effects
```

```
## [1] 0.943945
```

```
#Save Results in a Data Frame
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method="Movie Effects Model",
                                     RMSE = rmse_movie_effects))
# Clean up memory
rm(predicted_ratings)
gc()
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells 11289395 603.0   17424232  930.6  22137590 1182.3
## Vcells 94997609 724.8  321889858 2455.9 321889858 2455.9
```

3.1.3 Model with Both Movie and User Effects

Users can themselves have a personality bias while rating movies. Some users tend to give more positive or negative reviews than average because of their choice regardless of movie. To account for the variability of the users, another bias term can be included in the prediction as:

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

where: b_u = user-specific effect.

```
#Estimate User Effects
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

#Make predictions using Movie and User Effects
predicted_ratings_bi_bu <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

#Calculate RMSE of Model with Movie and User effects
rmse_movie_user_effects <- RMSE(predicted_ratings_bi_bu,
                                validation$rating)

#Save Results in a Data Frame
rmse_results <- rbind(rmse_results,
                      data_frame(Method = "Movie and User effect",
                                 RMSE = rmse_movie_user_effects))

# Clean up memory
rm(predicted_ratings_bi_bu, rmse_movie_user_effects)
gc()
```

```
##            used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells 11289444 603.0   17424232  930.6  22137590 1182.3
## Vcells 95102851 725.6  321889858 2455.9 321889858 2455.9
```

3.1.4 Model with Both Movie,User and Time Effects

Exploratory data analysis in the previous section also showed effect of time on the ratings of movies. The following prediction model incorporates the bias of time:

$$Y_{u,i} = \mu + b_i + b_u + f(d_{u,i}) + \varepsilon_{u,i}$$

where: $d_{u,i}$ = as the day for user's u rating of movie i and f denotes a smooth function.

```
#Create separate validation and training set with formatted timestamp
edx_time <- edx %>%
  mutate(timestamp = round_date(as_datetime(timestamp), unit = "week"))
validation_time <- validation %>%
  mutate(timestamp = round_date(as_datetime(timestamp), unit = "week"))
```

```
#Effect of time
time_avgs <- edx_time %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(timestamp) %>%
  summarize(b_t = mean(rating - mu - b_i - b_u))

#Make predictions using Movie, User and Time Effects
predicted_ratings_bi_bu_bt <- validation_time %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(time_avgs, by='timestamp') %>%
  mutate(pred = mu + b_i + b_u + b_t) %>%
  pull(pred)

#Calculate RMSE of Model with Both Movie,User and Time Effects
rmse_movie_user_time_effects <- RMSE(predicted_ratings_bi_bu_bt,
                                validation_time$rating)
rmse_movie_user_time_effects
```

```
## [1] 0.864989
```

```
#Save Results in a Data Frame
rmse_results <- rbind(rmse_results,
                    data_frame(Method = "Regression model using movie-user-date effect",
                               RMSE = rmse_movie_user_time_effects))

# Clean up memory
rm(predicted_ratings_bi_bu_bt, rmse_movie_user_time_effects)
gc()
```

```
##             used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  11289618 603.0   17424232  930.6  22137590 1182.3
## Vcells 105105220 801.9  321889858 2455.9 321889858 2455.9
```

3.2 Regularization Model

Some of the obscure movies in the database were rated by only a few users which introduces large uncertainty in estimating the bias of movie ratings. This can influence the overall prediction model and subsequently can increase our RMSE. Hence, the technique called regularization is applied which penalizes the loss function by adding a complexity term.

For the model method uses a tuning parmeter λ to define the penalty and minimise the RMSE. Modifying bi and bu for movies with limited ratings.

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

where: $d_{u,i}$ = as the day for user's u rating of movie i and f denotes a smooth function

When the sample size is very large, then the penalty term approaches zero and is ignored.

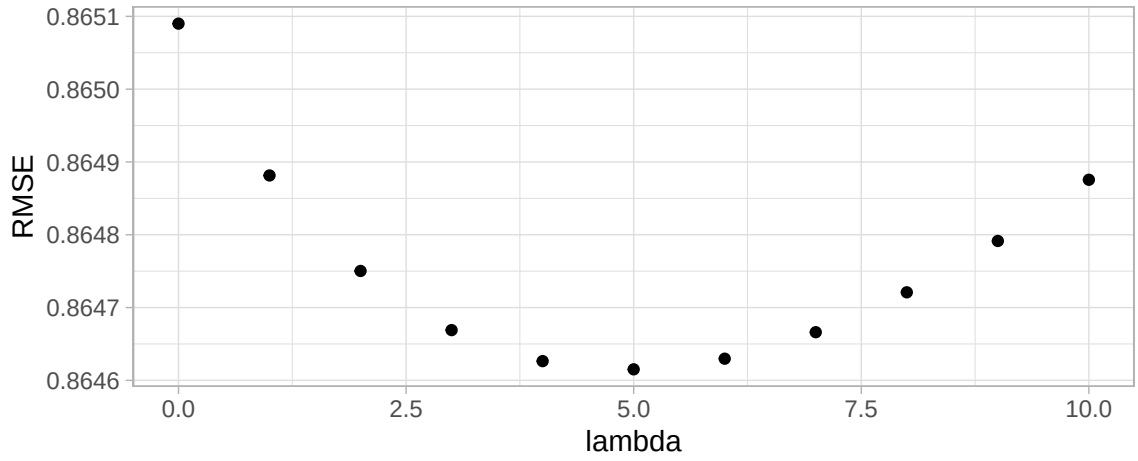
3.2.1 Regularization with Movie and User Effects

The first model using regularization in this project incorporates modified movie and user effect.

```r
#Define lambda values for cross-validation purpose
lambdas <- seq(0, 10, 1)

#Calculate RMSEs for each lambda
rmses <- sapply(lambdas, function(l){
  mu_reg <- mean(edx$rating)
  b_i_reg <- edx %>%
    group_by(movieId) %>%
    summarize(b_i_reg = sum(rating - mu_reg)/(n()+l))
  b_u_reg <- edx %>%
    left_join(b_i_reg, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u_reg = sum(rating - b_i_reg - mu_reg)/(n()+l))
  predicted_ratings_b_i_u <-
    validation %>%
    left_join(b_i_reg, by = "movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    mutate(pred = mu_reg + b_i_reg + b_u_reg) %>%
    pull(pred)
  return(RMSE(validation$rating,predicted_ratings_b_i_u))
})

#Plot RMSE vs lambda
ggplot(mapping = aes(x = lambdas, y = rmses)) + theme_light()+
  xlab("lambda") +
  ylab("RMSE") +
  geom_point()
```




```r
#Extract lambda that minimizes RMSE
lambda_min <- lambdas[which.min(rmses)]
lambda_min
```

```
## [1] 5
```

```r
#Extract minimized RMSE and save it into a dataframe
rmse_reg_movie_user_effects <- min(rmses)
rmse_results <- rbind(rmse_results,
                      data_frame(Method = "Regularization model using movie-user effect",
                                 RMSE = rmse_reg_movie_user_effects))
```

```
# Clean up memory
rm(lambdas,rmses)
gc()
```

```
##             used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  11307632 603.9   17424232  930.6  22137590 1182.3
## Vcells 105146662 802.3  321889858 2455.9 321889858 2455.9
```

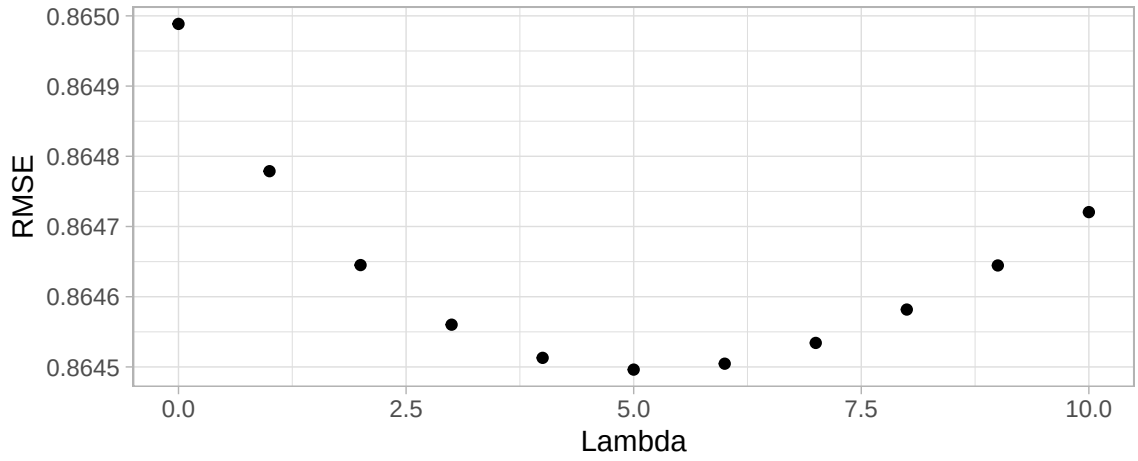
3.2.2 Regularization with Movie,User and Time Effects

The effects of time was also included in the regularization model.

```
#Define lambda values for cross-validation purpose
lambdas <- seq(0, 10, 1)

#Calculate RMSEs for each lambda
rmses <- sapply(lambdas, function(l){
  mu_reg <- mean(edx_time$rating)
  b_i_reg <- edx_time %>%
    group_by(movieId) %>%
    summarize(b_i_reg = sum(rating - mu_reg)/(n()+l))
  b_i_reg
  b_u_reg <- edx_time %>%
    left_join(b_i_reg, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u_reg = sum(rating - b_i_reg - mu_reg)/(n()+l))
  b_u_reg
  b_t_reg <- edx_time %>%
    left_join(b_i_reg, by="movieId") %>%
    left_join(b_u_reg, by="userId") %>%
    group_by(timestamp) %>%
    summarize(b_t_reg = sum(rating - b_i_reg - b_u_reg - mu_reg)/(n()+l))
    b_t_reg
  predicted_ratings <-
    validation_time %>%
    left_join(b_i_reg, by = "movieId") %>%
    left_join(b_u_reg, by = "userId") %>%
    left_join(b_t_reg, by = "timestamp") %>%
    mutate(pred = mu_reg + b_i_reg + b_u_reg + b_t_reg) %>%
    pull(pred)
  return(RMSE(validation$rating,predicted_ratings))
})

#Plot RMSE vs lambda
ggplot(mapping = aes(x = lambdas, y = rmses)) + theme_light()+
  xlab("Lambda") +
  ylab("RMSE") +
  geom_point()
```




```
#Extract lambda that minimizes RMSE
lambda_min <- lambdas[which.min(rmses)]
lambda_min
```

```
## [1] 5
```

```
#Extract minimized RMSE
rmse_reg_movie_user_effects <- min(rmses)
rmse_results <- rbind(rmse_results,
                      data_frame(Method = "Regularization model using movie,user and time effect",
                                 RMSE = rmse_reg_movie_user_effects))
# Clean up memory
rm(lambdas,rmses)
gc()
```

```
##             used  (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells  11308240 604.0   17424232  930.6  22137590 1182.3
## Vcells 105151653 802.3  321889858 2455.9 321889858 2455.9
```

3.3 Results

The RMSE values obtained for the different models implemented in this project are listed below:

```
#Print out summary of results
kable(rmse_results) %>%
  kable_styling(full_width = F)%>%
  row_spec(0, bold = T, color = "black", background = "#EBF2F6")
```

Method	RMSE
Naive Mean Baseline Model	1.061000
Movie Effects Model	0.943945
Movie and User effect	0.865090
Regression model using movie-user-date effect	0.864989
Regularization model using movie-user effect	0.864615
Regularization model using movie,user and time effect	0.864496

The results showed that regularization technique had the best performance when taking into account multiple effects of bias (such as user or movie specific) and the varibility due to time.

4 Conclusions

In this project the MovieLens 10M dataset has been used to build several recommendation system algorithms to predict movie ratings of a given user. The training set and test set have been provided by the Edx course with and linear regression models and regularization technique have been applied step-by-step to improve upon the mean-baseline model performance. All of the models are evaluated against RMSE (root mean squared error) metric defined in the course text book Introduction to Data Science by Prof. Rafael A. Irizarry. Overall the regularization technique using movie-user effect and movie,user and time effect have resulted in an RMSE less than 0.8649.

An interesting prospect would future work would be to apply Cross-validation to improve the prediction model and apply other methods such as Matrix Factorization.

The project is uploaded in my GitHub repository and can be accessed through this link.

5 References

[1] Irizzary,R., 2018,Introduction to Data Science,github page,https://rafalab.github.io/dsbook/