HarvardX - PH125.9x Data Science: Capstone - MovieLens Project

NATT SUTTHIKULKARN

6/23/2020

1 Overview

1.1 Introduction

Recommendation systems are one of the most interesting applications of Machine Learning and platforms such as Netflix and YouTube use them extensively to recommend movies and videos that we may want to watch. Creating recommendation systems involves identifying features that helps to predict the rating any given user will give to any movie.

We begin with a simple model, which uses the mean of the observed values. In addition, the user, movie and even time effects are incorporated into the linear model. Next, we investigate the regularization of the user and movie effects that penalizes samples with few ratings before trying a method that is used widely called Matrix Factorization.

1.2 Project Goals

To evaluate machine learning algorithms, we determine the loss function which measures the difference between the predicted value and the actual outcome. The error metric central to this project is the root mean squared error (RMSE). Ultimately, we need to create a recommendation system with RMSE lower than 0.8649.

For the chosen model that yields RMSE close to or below the target with test dataset, we eventually train the edx dataset with that particular model and use the validation set for final validation. The validation should not be used to compare different models to each other and should ONLY be used for evaluating the RMSE of the final algorithm.

Data visualization is applied to gain insights and understand how the features can impact the outcome and help to build the machine learning model.

1.3 Dataset

Firstly, we download the MovieLens dataset and split into two subsets used for training and validation. The training set is called edx and the validation set is called validation. The edx set is further split into two subsets used for training and and testing.

```
################################## 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 --
```

```
## v ggplot2 3.3.1     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- 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-project.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
```

```r
# 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(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")
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")
```

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

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

```r
# Loading library
library(tidyverse)
library(caret)
```

```r
library(data.table)
library(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 objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

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

```
## Loading required package: kableExtra
```

```
##
## Attaching package: 'kableExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     group_rows
```

```r
library(kableExtra)
```

2 Methods/ Analysis

2.1 Methods

Machine learning models performed:

1. Prediction using mean rating
2. Prediction using movie effect on mean rating
3. Prediction using movie and user effects on mean rating
4. Prediction using movie, user and time effects on mean rating
5. Prediction using movie and user effects on mean rating and regularizing the model with optimal lambda
6. Prediction using matrix factorization with gradient descent

2.2 Data Exploration and Visualization

The dataset contains six columns (i.e., "userID", "movieID", "rating", "timestamp", "title", and "genres"), with each row representing a single rating of a user for a single movie.

```
kable(head(edx), "pandoc", caption = "edx dataset")
```

Table 1: edx dataset

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	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

There are 9,000,055 rows in total with movies rated between 0.5 and 5.0.

```
kable(summary(edx), "pandoc", caption = "edx summary")
```

Table 2: edx summary

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

Within edx, you will find ~10,700 distinct userId, ~70,000 distinct movieId and ~800 distinct genre combinations with ratings mean of ~3.5 as follows:

```
distinct <- edx %>% summarize(n_users = n_distinct(userId),
    n_movies = n_distinct(movieId), n_genres = n_distinct(genres))
kable(distinct, "pandoc", caption = "Distinct number of userID, movieId and genres")
```

Table 3: Distinct number of userID, movieId and genres

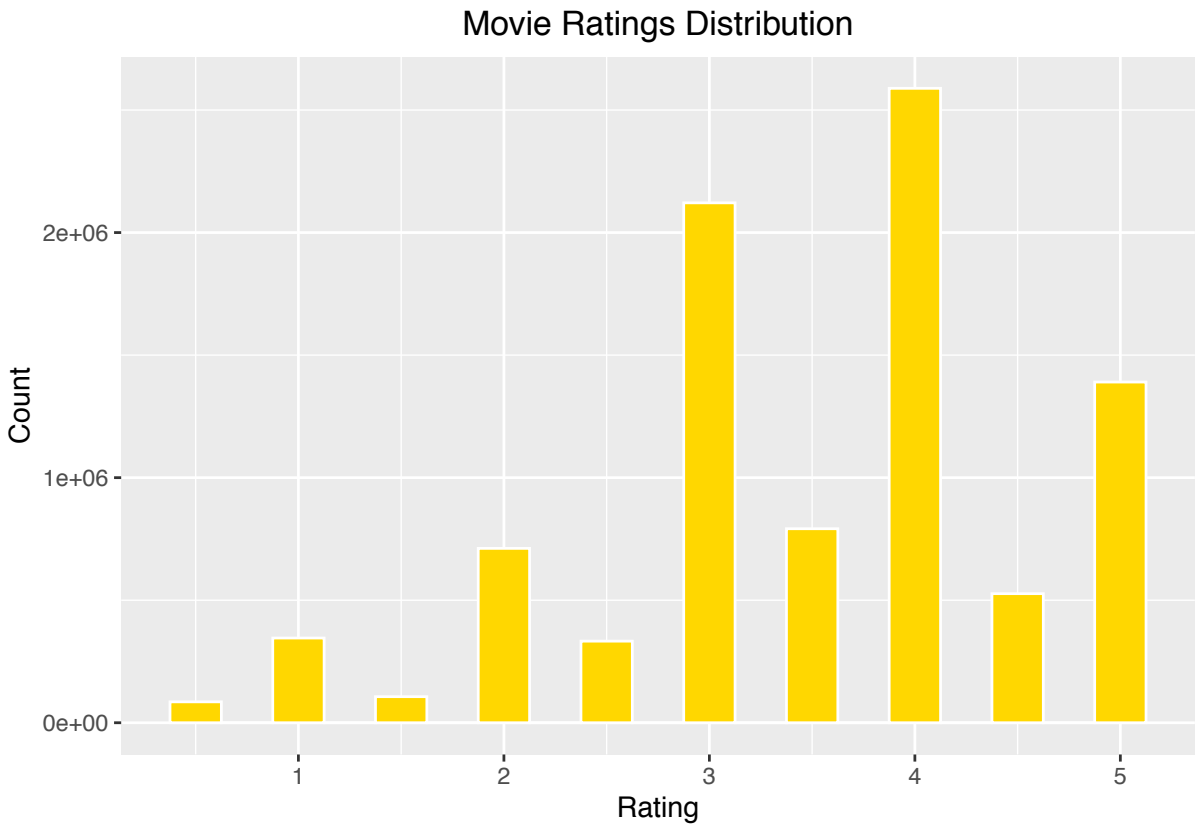
n_users	n_movies	n_genres
69878	10677	797

```
rating_mean <- mean(edx$rating)  # Ratings Mean
rating_mean
```

```
## [1] 3.512465
```

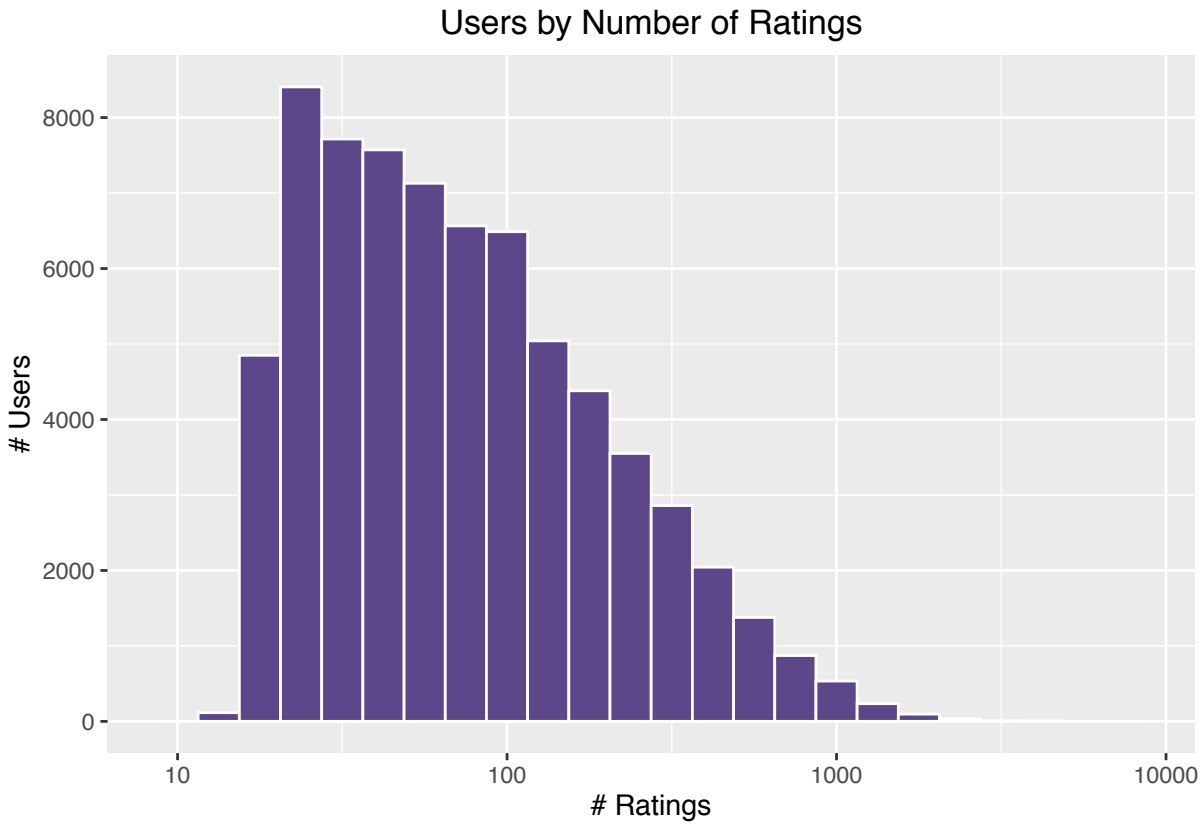
Distribution of movie ratings in edx are shown, in descending order of 4.0, 3.0 and 5.0 etc. Users tend to give whole ratings more often than half ratings.

```
edx %>% ggplot(aes(rating)) + geom_histogram(binwidth = 0.25,
    fill = "gold", color = "white") + xlab("Rating") +
    ylab("Count") + ggtitle("Movie Ratings Distribution") +
    theme(plot.title = element_text(hjust = 0.5))  # center the title
```



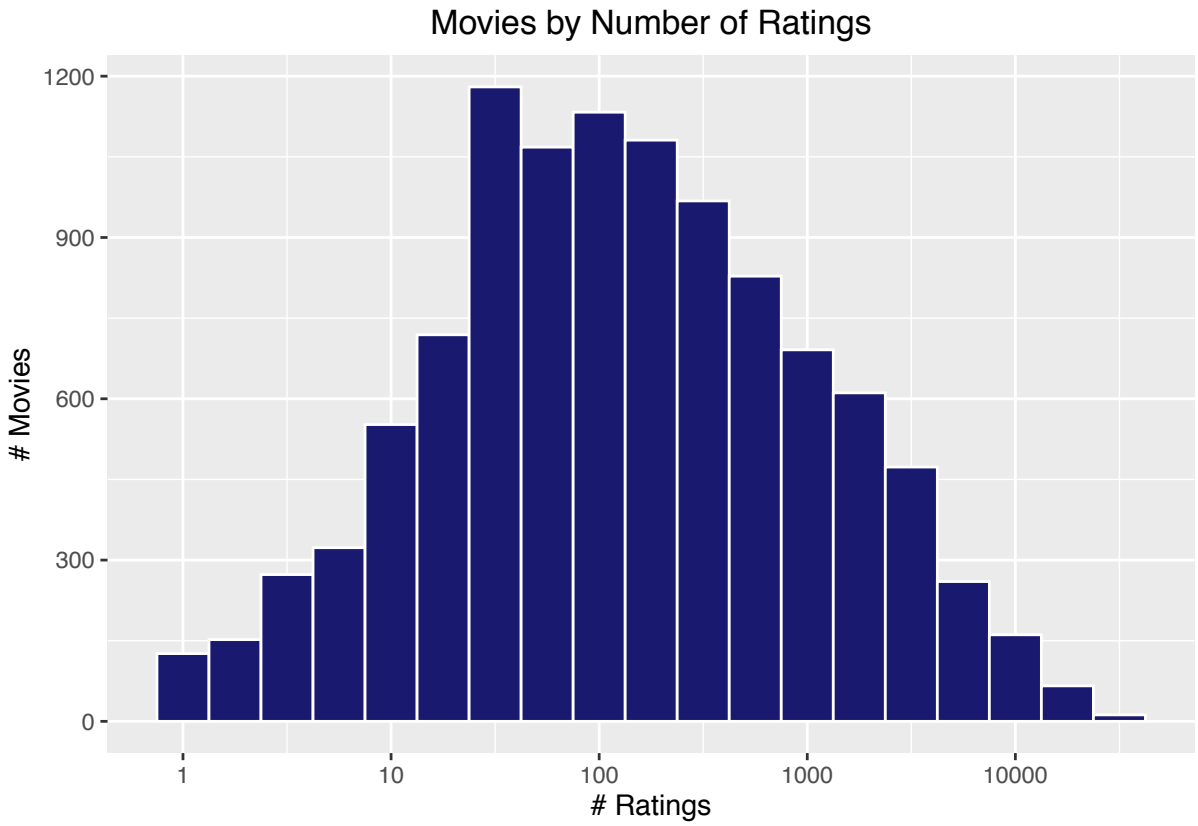
Now let's explore how often a user rates a movie. In most cases, users rate between 10 to 100 movies and over 1,000 movies in some cases.

```
edx %>% count(userId) %>% ggplot(aes(n)) + geom_histogram(binwidth = 0.125,
    fill = "mediumpurple4", color = "white") + scale_x_log10() +
    xlab("# Ratings") + ylab("# Users") + ggtitle("Users by Number of Ratings") +
    theme(plot.title = element_text(hjust = 0.5))
```



Distribution for number of ratings per movie indicates that there are movies rated less than 10 times which could make predicting future ratings less accurate.

```
edx %>% count(movieId) %>% ggplot(aes(n)) + geom_histogram(binwidth = 0.25,
    fill = "midnightblue", color = "white") + scale_x_log10() +
    xlab("# Ratings") + ylab("# Movies") + ggtitle("Movies by Number of Ratings") +
    theme(plot.title = element_text(hjust = 0.5))
```



2.3 Effects by Model

Movie Effects Next we calculate the average rating per movie and the rating frequency.

```
avg_movie <- edx %>% group_by(title) %>% summarize(count_movie_rating = n(),
    avg_movie_rating = mean(rating), .groups = "drop") %>%
    arrange(desc(count_movie_rating))
kable(head(avg_movie), "pandoc", caption = "Example of rating frequency and average rating
```

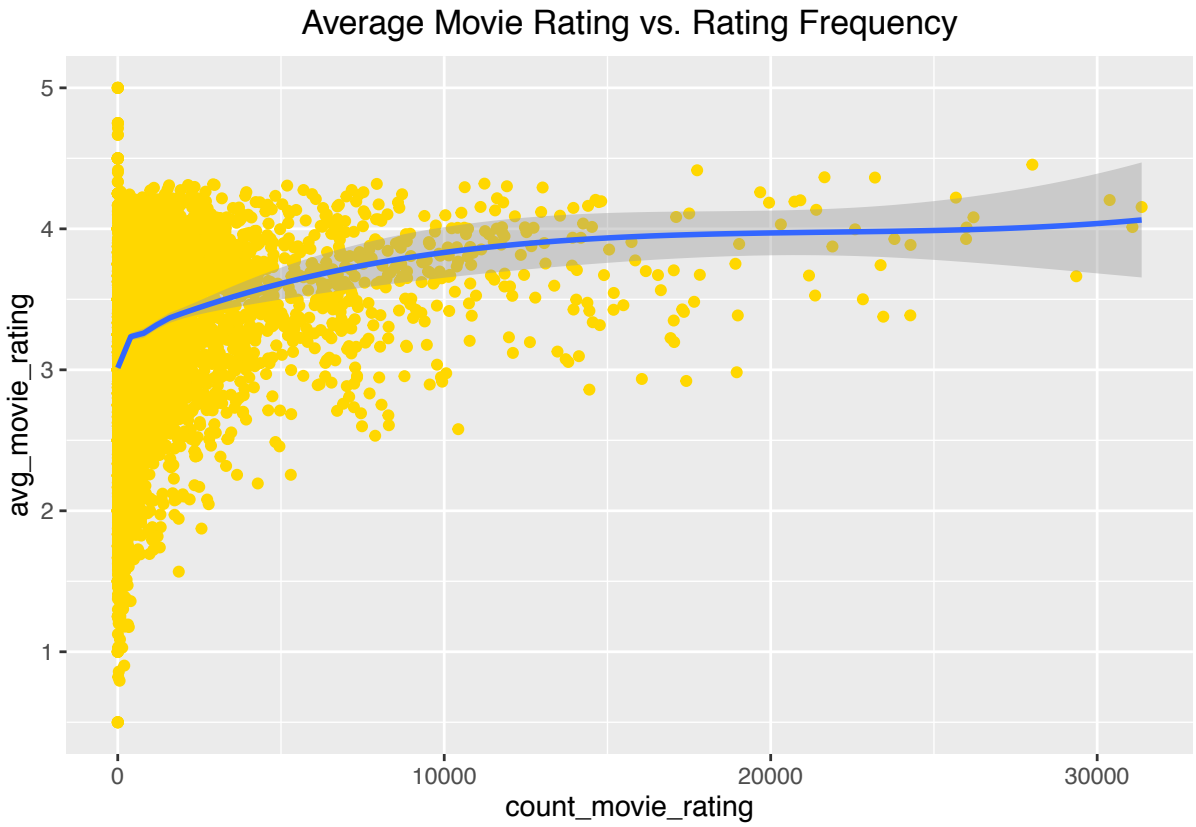
Table 4: Example of rating frequency and average rating received per movie

title	count_movie_rating	avg_movie_rating
Pulp Fiction (1994)	31362	4.154789
Forrest Gump (1994)	31079	4.012822
Silence of the Lambs, The (1991)	30382	4.204101
Jurassic Park (1993)	29360	3.663522
Shawshank Redemption, The (1994)	28015	4.455131
Braveheart (1995)	26212	4.081852

We can see that the average rating variation is relatively greater for movies that have been rated less often.

```
avg_movie %>% ggplot(aes(count_movie_rating, avg_movie_rating)) +
    geom_point(color = "gold") + geom_smooth(method = "loess") +
    ggtitle("Average Movie Rating vs. Rating Frequency") +
    theme(plot.title = element_text(hjust = 0.5))
```

```
## `geom_smooth()` using formula 'y ~ x'
```



User Effects Among users who rated over 100 movies, we can see the average rating varies between ~2.4 to ~3.8 for those who have given over 4,000 ratings.

```
avg_user <- edx %>% group_by(userId) %>% summarize(count_user_rating = n(),
    avg_user_rating = mean(rating), .groups = "drop") %>%
    filter(count_user_rating > 100) %>% arrange(desc(count_user_rating))
kable(head(avg_user), "pandoc", caption = "Example of rating frequency and average rating gi
```

Table 5: Example of rating frequency and average rating given per user

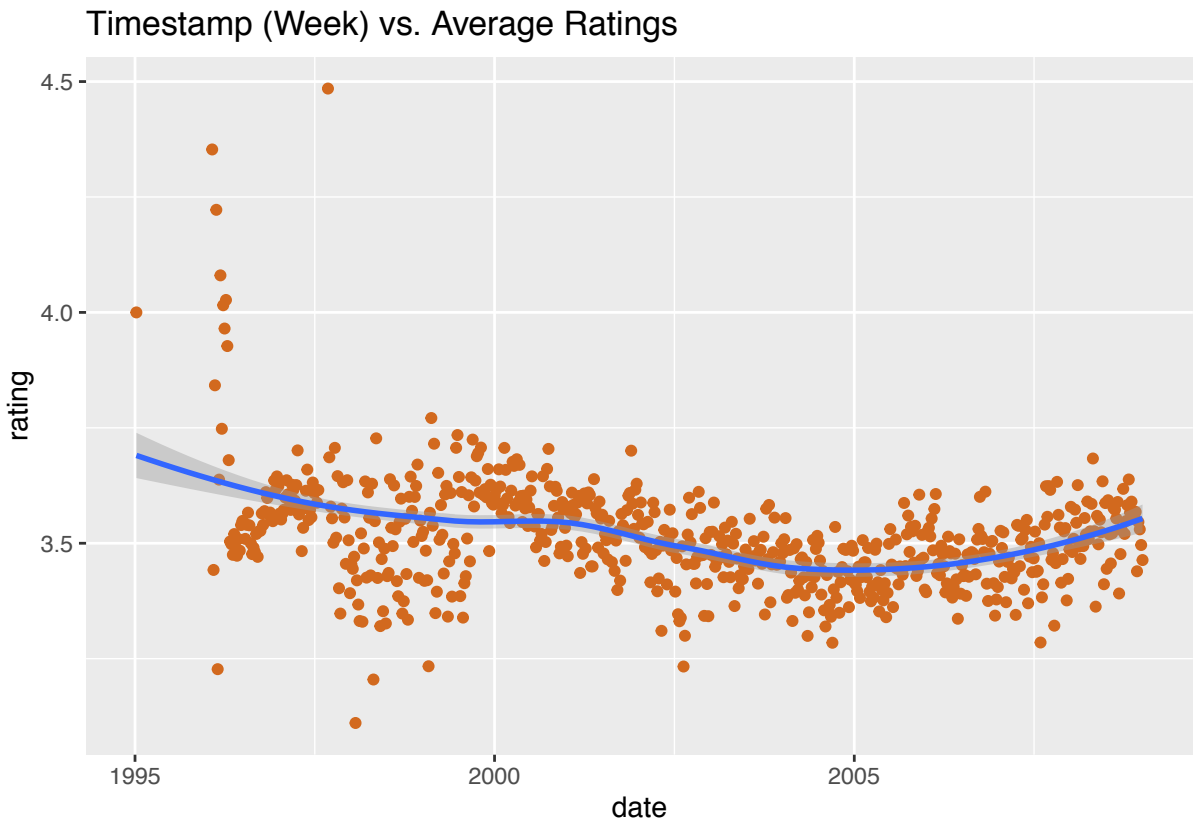
userId	count_user_rating	avg_user_rating
59269	6616	3.264586
67385	6360	3.197720

userId	count_user_rating	avg_user_rating
14463	4648	2.403615
68259	4036	3.576933
27468	4023	3.826871
19635	3771	3.498807

Time Effects The dataset includes a time stamp which represents the time and data in which the rating was provided. We convert time stamp to a 'datetime' object and round it to the weekly unit.

```
edx %>% mutate(date = round_date(as_datetime(timestamp),
    unit = "week")) %>% group_by(date) %>% summarize(rating = mean(rating),
    .groups = "drop") %>% ggplot(aes(date, rating)) +
    geom_point(color = "chocolate") + geom_smooth() +
    ggtitle("Timestamp (Week) vs. Average Ratings")
```

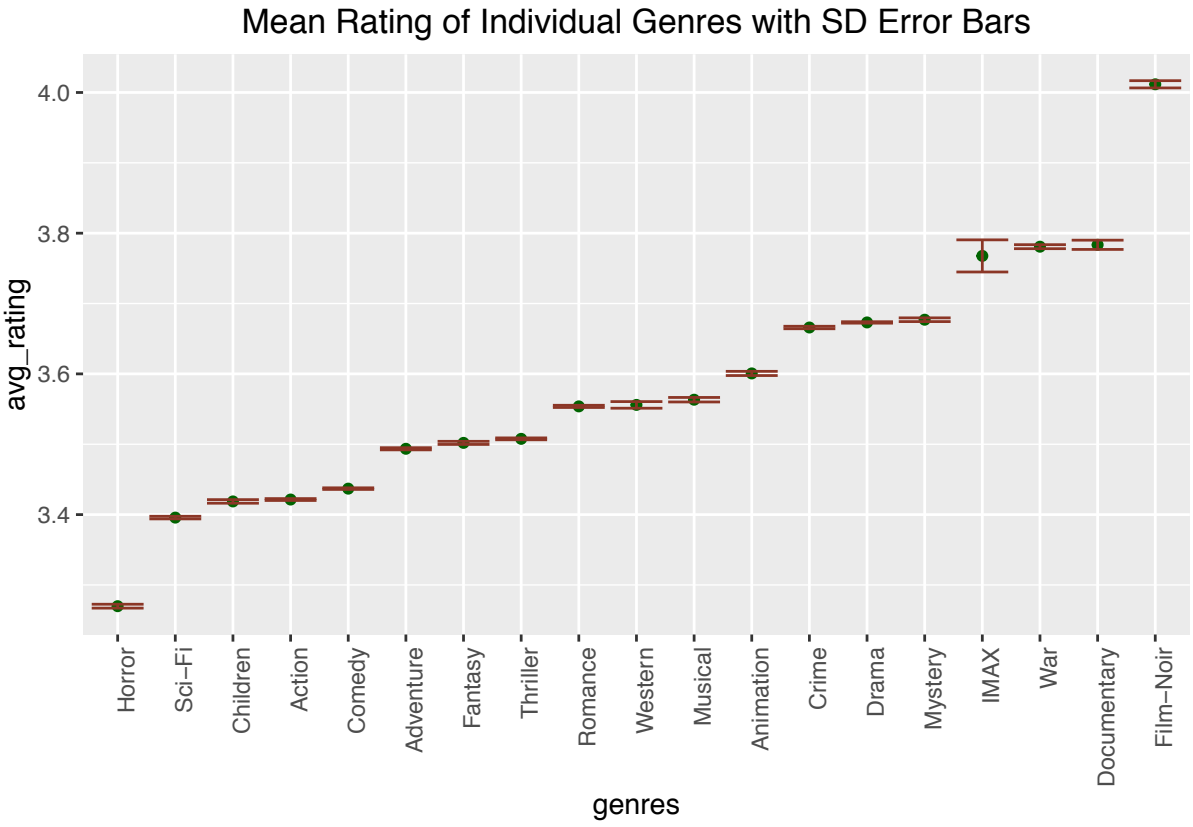
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



We can see that there is some evidence of a time effect in the plot, but there is not a strong effect of time.

Genre Effects We can deduce from the error bar plots that the highest and lowest average ratings for individual genre are Film-Noir and Horror respectively.

```
edx %>% separate_rows(genres, sep = "\\|") %>% group_by(genres) %>%
    summarize(n = n(), avg_rating = mean(rating), se = sd(rating)/sqrt(n()),
        .groups = "drop") %>% filter(n >= 1000) %>%
    mutate(genres = reorder(genres, avg_rating)) %>%
    ggplot(aes(x = genres, y = avg_rating, ymin = avg_rating -
        2 * se, ymax = avg_rating + 2 * se)) + geom_point(color = "darkgreen") +
    geom_errorbar(color = "tomato4") + ggtitle("Mean Rating of Individual Genres with SD Er
    theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
    theme(plot.title = element_text(hjust = 0.5))
```



3 Results and Discussion

The edx dataset is split into test and train sets according to the course instructions provided.

```
edx_test_index <- createDataPartition(y = edx$rating,
    times = 1, p = 0.2, list = FALSE)
train_set <- edx[-edx_test_index, ]
test_set <- edx[edx_test_index, ]
```

To compare different models or to see how well we're doing compared to a baseline, we will use root mean squared error (RMSE) as our loss function. We can interpret RMSE similar to standard deviation and it is defined as follows.

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

where N is the number of users, movie ratings, and the sum incorporating the total combinations.

```
RMSE <- function(true_ratings, predicted_ratings) {
    sqrt(mean((true_ratings - predicted_ratings)^2,
        na.rm = TRUE))
}
```

3.1 Base Model Using Mean Rating

This model assumes the same rating for all movies and all users, with all the differences explained by random variation and calculates the mean of the dataset to predict the movie ratings. The base model is represented by the following formula:

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

where $Y_{u,i}$ is the prediction, $\epsilon_{u,i}$ is the independent error, and μ the expected "true" rating for all movies. Predicting the mean gives the following RMSE.

```
mu_hat <- mean(train_set$rating)
naive_rmse <- RMSE(test_set$rating, mu_hat)
print(c("The Base Model RMSE :", naive_rmse))
```

```
## [1] "The Base Model RMSE :" "1.06070788648804"
```

```
rmse_results <- tibble(method = "Base Model - Mean Rating",
    RMSE_train_set = naive_rmse, RMSE_validation = "NA")
kable((rmse_results[1:1, ]), "pandoc", digits = 7,
    caption = "RMSE Model Results", align = "c")
```

Table 6: RMSE Model Results

method	RMSE_train_set	RMSE_validation
Base Model - Mean Rating	1.060708	NA

RMSE result for Base Model is not ideal.

3.2 Movie Effects Model

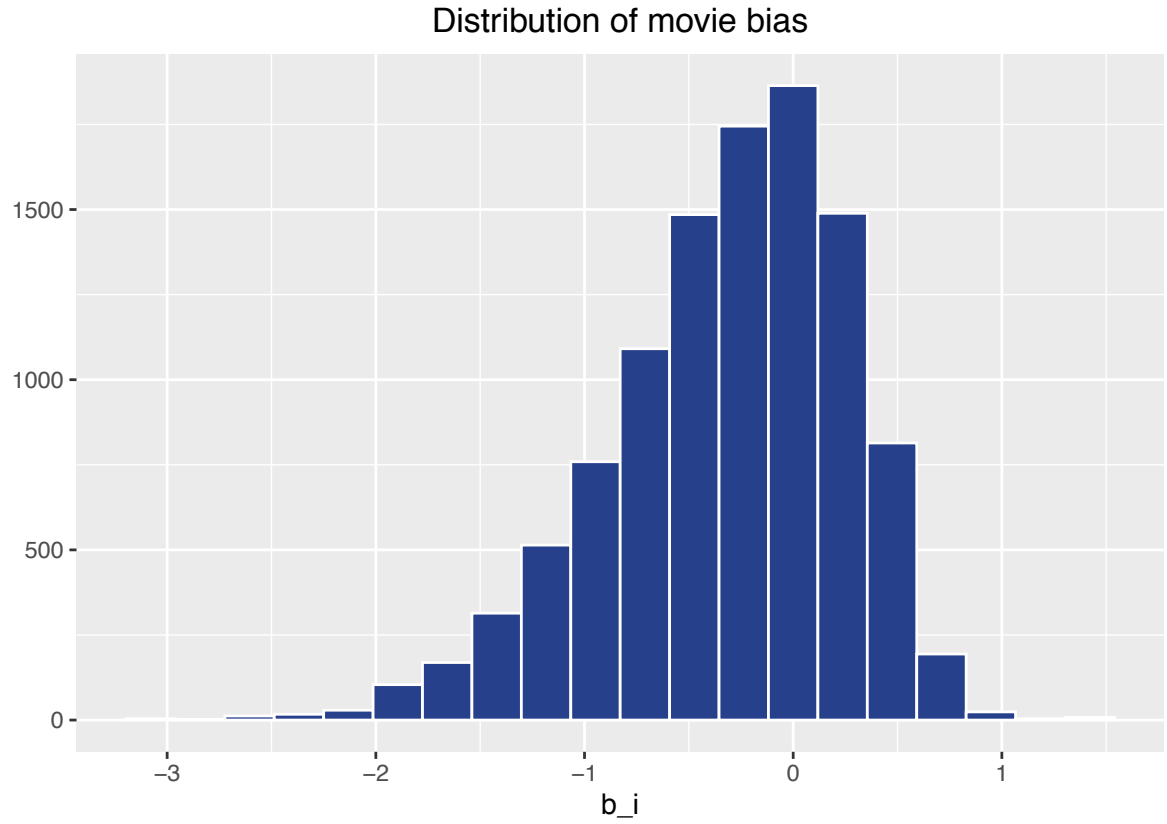
The model takes into account a bias term for each movie based on the difference between the movies mean rating and the overall mean rating since popular movies are rated relatively higher than unpopular ones. This bias is calculated as follows:

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

where $Y_{u,i}$ is the prediction, $\epsilon_{u,i}$ is the independent error, and μ the mean rating for all movies, and b_i is the bias for each movie i.

We can see the distribution of movie bias in the plot below.

```
mu <- mean(train_set$rating)
movie_avgs <- train_set %>% group_by(movieId) %>% summarize(b_i = mean(rating -
    mu), .groups = "drop")

qplot(b_i, data = movie_avgs, bins = 20, color = I("white"),
    fill = I("royalblue4"), main = "Distribution of movie bias") +
    theme(plot.title = element_text(hjust = 0.5))
```



We can improve our movie rating prediction by adding the computed b_i to μ within the model.

```
predicted_ratings <- mu + test_set %>% left_join(movie_avgs,
    by = "movieId") %>% pull(b_i)

rmse_movies <- RMSE(test_set$rating, predicted_ratings)
print(c("The Movie Effects Model RMSE :", rmse_movies))
```

```
## [1] "The Movie Effects Model RMSE :" "0.943714419701666"
```

```
rmse_results <- add_row(rmse_results, method = "Movie Effects Model",
    RMSE_train_set = rmse_movies, RMSE_validation = "NA")
kable((rmse_results[1:2, ]), "pandoc", digits = 7,
    caption = "RMSE Model Results", align = "c")
```

Table 7: RMSE Model Results

method	RMSE_train_set	RMSE_validation
Base Model - Mean Rating	1.0607079	NA
Movie Effects Model	0.9437144	NA

The Movie Effects Model utlizing both movie bias, b_i, and mean, μ, gives an improved prediction with a lower RMSE value i.e. 0.9437144.

3.3 Movie and User Effects Model

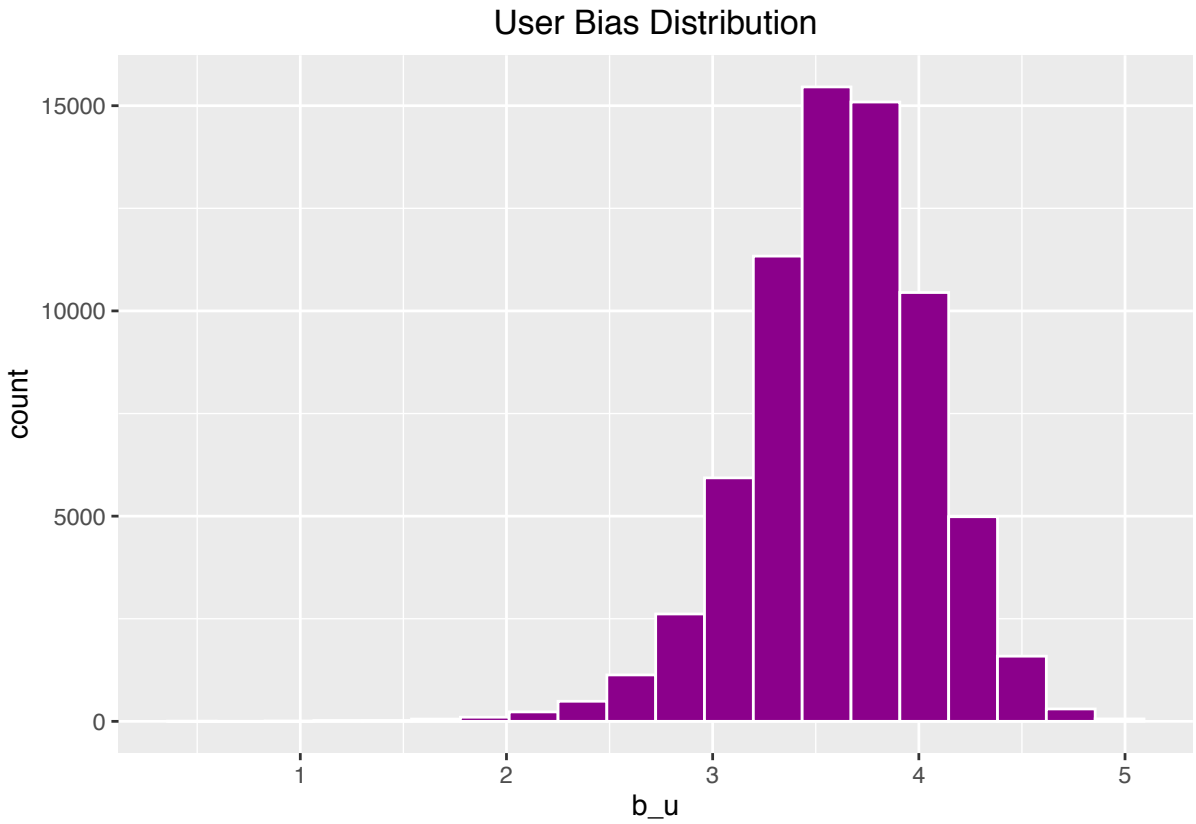
By incorporating the individual user effects, b_u, we can account for each user's inherent bias to rate all films higher or lower.

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

where $Y_{u,i}$ is the prediction, $\epsilon_{u,i}$ is the independent error, and μ the mean rating for all movies, b_i is the bias for each movie i, and b_u is the bias for each user u.

Let's compute the average rating for user u for those that have rated over 100 movies:

```
train_set %>% group_by(userId) %>% summarize(b_u = mean(rating),
    .groups = "drop") %>% filter(n() >= 100) %>% ggplot(aes(b_u)) +
    geom_histogram(bins = 20, color = "white", fill = "magenta4") +
    ggtitle("User Bias Distribution") + theme(plot.title = element_text(hjust = 0.5))
```



Let's predict and obtain RMSE using the movie and user effects model.

```
user_avgs <- train_set %>% left_join(movie_avgs, by = "movieId") %>%
    group_by(userId) %>% summarize(b_u = mean(rating -
    mu - b_i), .groups = "drop")

predicted_ratings_bu <- test_set %>% left_join(movie_avgs,
    by = "movieId") %>% left_join(user_avgs, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>% pull(pred)

rmse_movies_users <- RMSE(test_set$rating, predicted_ratings_bu)

rmse_results <- add_row(rmse_results, method = "Movie & User Effects Model",
    RMSE_train_set = rmse_movies_users, RMSE_validation = "NA")
kable((rmse_results[1:3, ]), "pandoc", digits = 7,
    caption = "RMSE Results", align = "c")
```

Table 8: RMSE Results

method	RMSE_train_set	RMSE_validation
Base Model - Mean Rating	1.0607079	NA
Movie Effects Model	0.9437144	NA
Movie & User Effects Model	0.8661625	NA

By incorporating user-specific bias into this model leads to further reduction of RMSE i.e. 0.8661625.

3.4 Movie-User-Time Effects Model

Within data exploration, we investigate time effect where we define $d_{u,i}$ as the day for user's u rating of movie i. The new model is as follows:

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

where f is a smooth function of $d_{u,i}$.

A further and detailed explanation of time effects are developed in Koren(2009).

Let's predict and obtain RMSE using the movie, user and time effects model.

```
time_avgs <- train_set %>% left_join(movie_avgs, by = "movieId") %>%
    left_join(user_avgs, by = "userId") %>% mutate(date = round_date(as_datetime(timestamp)
    unit = "week")) %>% group_by(date) %>% summarize(b_t = mean(rating -
    mu - b_i - b_u), .groups = "drop")

predicted_ratings_bt <- test_set %>% left_join(movie_avgs,
    by = "movieId") %>% left_join(user_avgs, by = "userId") %>%
    mutate(date = round_date(as_datetime(timestamp),
        unit = "week")) %>% left_join(time_avgs, by = "date") %>%
    mutate(pred = mu + b_i + b_u + b_t) %>% pull(pred)

rmse_movie_user_time <- RMSE(test_set$rating, predicted_ratings_bt)

rmse_results <- add_row(rmse_results, method = "Movie + User + Time Effects Model",
    RMSE_train_set = rmse_movie_user_time, RMSE_validation = "NA")
kable((rmse_results[1:4, ]), "pandoc", digits = 7,
    caption = "RMSE Results", align = "c")
```

Table 9: RMSE Results

method	RMSE_train_set	RMSE_validation
Base Model - Mean Rating	1.0607079	NA
Movie Effects Model	0.9437144	NA
Movie & User Effects Model	0.8661625	NA
Movie + User + Time Effects Model	0.8660880	NA

By incorporating time-specific feature into this model leads to a relatively insignificant reduction of RMSE by less than 0.01% i.e. 0.8660880.

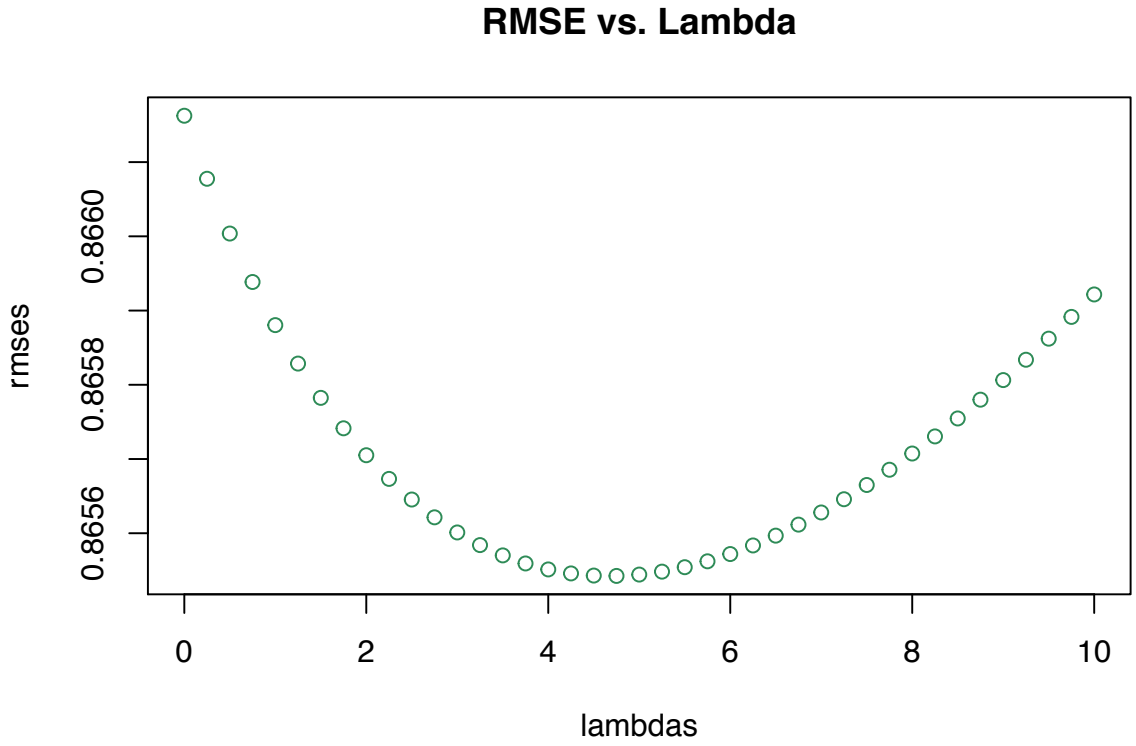
3.5 Regularized Movie-User Effects Model

Regularization stems errors attributed by movies with very few ratings and users who rate less frequently. We can improve the predictions by applying regularization to reduce known uncertainty which can skew the error metric. This method uses a tuning parameter, λ, to minimize the RMSE when large errors are particularly undesirable. Modifying b_i and b_u for movies constrained by ratings.

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

Determine the tuning parameter (lambda) by plotting RMSE vs. lambda.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l) {

    mu <- mean(train_set$rating)

    b_i <- train_set %>% group_by(movieId) %>% summarize(b_i = sum(rating -
        mu)/(n() + l), .groups = "drop")

    b_u <- train_set %>% left_join(b_i, by = "movieId") %>%
        group_by(userId) %>% summarize(b_u = sum(rating -
        b_i - mu)/(n() + l), .groups = "drop")

    predicted_ratings_lam <- test_set %>% left_join(b_i,
        by = "movieId") %>% left_join(b_u, by = "userId") %>%
        mutate(pred = mu + b_i + b_u) %>% pull(pred)

    return(RMSE(predicted_ratings_lam, test_set$rating))
})

plot(lambdas, rmses, main = "RMSE vs. Lambda", col = "seagreen4")
```



Obtain optimal lambda that minimizes RMSE from the plot.

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

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

Next we obtain RMSE from regularized user and movie effects.

```
rmse_movie_user_reg <- min(rmses)

rmse_results <- add_row(rmse_results, method = "Regularized Movie + User Effects Model",
    RMSE_train_set = rmse_movie_user_reg, RMSE_validation = "NA")
rmse_results
```

```
kable((rmse_results[1:5, ]), "pandoc", digits = 7,
    caption = "RMSE Results", align = "c")
```

Table 10: RMSE Results

method	RMSE_train_set	RMSE_validation
Base Model - Mean Rating	1.0607079	NA
Movie Effects Model	0.9437144	NA

method	RMSE_train_set	RMSE_validation
Movie & User Effects Model	0.8661625	NA
Movie + User + Time Effects Model	0.8660880	NA
Regularized Movie + User Effects Model	0.8655425	NA

By regularization of user and movie effects, we further reduce RMSE to 0.8655425. However, this is still ~0.07% above the target RMSE of 0.8649.

3.6 Matrix Factorization with GD

To improve the RMSE, we will apply Matrix Factorization with parallel stochastic gradient descent utilizing recosystem - an R wrapper of the LIBMF library which creates a Recommender System by Using Parallel Matrix Factorization (Chin, Yuan, et al. 2015). The main task of recommender system is to predict unknown entries in the rating matrix based on observed values.

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

```
## Loading required package: recosystem
```

```
set.seed(123, sample.kind = "Rounding")  # a randomized algorithm
```

```
## Warning in set.seed(123, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
library(recosystem)
```

According to Qiu(2020), recosystem is done in the following applicable steps:

1. Create a model object by calling Reco().
2. Call the $tune() method to select best tuning parameters along a set of candidate values.
3. Train the model by calling the $train() method where tuning parameters can be set inside the function
4. Use the $predict() method to compute predicted values.

```
# Convert the train and test sets into recosystem
# input format
train_reco <- with(train_set, data_memory(user_index = userId,
    item_index = movieId, rating = rating))
test_reco <- with(test_set, data_memory(user_index = userId,
    item_index = movieId, rating = rating))
```

```
# Create the model object
r <- recosystem::Reco()
```

Determine optimal tuning parameters 'opts'

```
# opts <- r$tune(train_reco, opts = list(dim =
# c(10, 20, 30), lrate = c(0.1, 0.2), costp_l2 =
# c(0.01, 0.1), costq_l2 = c(0.01, 0.1), nthread =
# 4, niter = 10))
```

For details of tuning parameters, access https://cran.r-project.org/web/packages/recosystem/recosystem.pdf

In order to avoid performance issues, the 'opts' chunk has been pre-run to derive the optimal tuning parameters as per below:

```
opts <- r$tune(train_reco, opts = list(dim = c(10),
    lrate = c(0.1), costp_l2 = c(0.1), costq_l2 = c(0.1),
    nthread = 4, niter = 10, verbose = TRUE))
```

```
## ============================
## dim     : 10
## costp_l1: 0
## costp_l2: 0.1
## costq_l1: 0
## costq_l2: 0.1
## lrate   : 0.1
##
## fold      rmse
##    0    0.8408
##    1    0.8398
##    2    0.8418
##    3    0.8413
##    4    0.8394
## ==============
##  avg    0.8406
## ============================
##
## ============================
## dim     : 10
## costp_l1: 0.1
## costp_l2: 0.1
## costq_l1: 0
## costq_l2: 0.1
## lrate   : 0.1
##
## fold      rmse
```

```
##    0    0.8746
##    1    0.8684
##    2    0.8775
##    3    0.8756
##    4    0.8777
## ==============
##  avg    0.8748
## ============================
##
## ============================
## dim     : 10
## costp_l1: 0
## costp_l2: 0.1
## costq_l1: 0.1
## costq_l2: 0.1
## lrate   : 0.1
##
## fold      rmse
##    0    0.8720
##    1    0.8754
##    2    0.8733
##    3    0.8801
##    4    0.8742
## ==============
##  avg    0.8750
## ============================
##
## ============================
## dim     : 10
## costp_l1: 0.1
## costp_l2: 0.1
## costq_l1: 0.1
## costq_l2: 0.1
## lrate   : 0.1
##
## fold      rmse
##    0    0.8698
##    1    0.8897
##    2    0.8890
##    3    0.8840
##    4    0.8853
## ==============
##  avg    0.8836
## ============================
```

```
# Train the algorithm
r$train(train_reco, opts = c(opts$min, nthread = 4,
    niter = 10))
```

```
## iter      tr_rmse          obj
##    0       0.9709   1.1934e+07
##    1       0.8831   1.0674e+07
##    2       0.8687   1.0589e+07
##    3       0.8500   1.0398e+07
##    4       0.8431   1.0329e+07
##    5       0.8394   1.0297e+07
##    6       0.8366   1.0275e+07
##    7       0.8343   1.0259e+07
##    8       0.8325   1.0245e+07
##    9       0.8309   1.0235e+07
```

```
# Calculate the predicted values (ratings)
y_hat_reco <- r$predict(test_reco, out_memory())
head(y_hat_reco, 10)
```

```
##  [1] 4.029479 4.903122 3.022567 3.462974 3.793452 3.491639 2.919573 4.101493
##  [9] 3.923182 3.495960
```

```
# Matrix Factorization - recosystem
rmse_matfac <- RMSE(test_set$rating, y_hat_reco)
print(c("The Matrix Factorization RMSE :", rmse_matfac))
```

```
## [1] "The Matrix Factorization RMSE :" "0.84185287378031"
```

```
# Add new results to tibble for comparison
rmse_results <- add_row(rmse_results, method = "Matrix Factorization GD Model",
    RMSE_train_set = rmse_matfac, RMSE_validation = "NA")
rmse_results
```

```
kable((rmse_results[1:6, ]), "pandoc", digits = 7,
    caption = "RMSE Results", align = "c")
```

Table 11: RMSE Results

method	RMSE_train_set	RMSE_validation
Base Model - Mean Rating	1.0607079	NA
Movie Effects Model	0.9437144	NA
Movie & User Effects Model	0.8661625	NA
Movie + User + Time Effects Model	0.8660880	NA
Regularized Movie + User Effects Model	0.8655425	NA
Matrix Factorization GD Model	0.8418529	NA

Through matrix factorization model, we obtain RMSE which is below the target of 0.8649. This

is by far the lowest RMSE obtained. Hence, Matrix Factorization is the chosen model for the final validation using validation dataset.

3.7 Final Validation

The project goal is achieved if the RMSE stays below the target. Based on the RMSE Results table, matrix factorization achieves the target RMSE. For the final RMSE calculation, we use edx and validation as train and test sets respectively to determine RMSE with the validation set.

```
set.seed(123, sample.kind = "Rounding")
```

Matrix Factorization Model RMSE

```
## Warning in set.seed(123, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
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))
r <- recosystem::Reco()
```

Call the $tune() method apply the best tuning parameters.

```
opts <- r$tune(edx_reco, opts = list(dim = c(10), lrate = c(0.1),
    costp_l2 = c(0.1), costq_l2 = c(0.1), nthread = 4,
    niter = 10, verbose = TRUE))
```

```
## ============================
## dim     : 10
## costp_l1: 0
## costp_l2: 0.1
## costq_l1: 0
## costq_l2: 0.1
## lrate   : 0.1
##
## fold      rmse
##    0    0.8373
##    1    0.8331
##    2    0.8353
##    3    0.8378
##    4    0.8358
## ==============
```

```
##  avg    0.8359
## ============================
##
## ============================
## dim     : 10
## costp_l1: 0.1
## costp_l2: 0.1
## costq_l1: 0
## costq_l2: 0.1
## lrate   : 0.1
##
## fold      rmse
##    0    0.8719
##    1    0.8662
##    2    0.8741
##    3    0.8745
##    4    0.8732
## ==============
##  avg    0.8720
## ============================
##
## ============================
## dim     : 10
## costp_l1: 0
## costp_l2: 0.1
## costq_l1: 0.1
## costq_l2: 0.1
## lrate   : 0.1
##
## fold      rmse
##    0    0.8736
##    1    0.8781
##    2    0.8682
##    3    0.8716
##    4    0.8779
## ==============
##  avg    0.8739
## ============================
##
## ============================
## dim     : 10
## costp_l1: 0.1
## costp_l2: 0.1
## costq_l1: 0.1
## costq_l2: 0.1
## lrate   : 0.1
##
## fold      rmse
```

```
##    0    0.8885
##    1    0.8813
##    2    0.8661
##    3    0.8841
##    4    0.8847
## ==============
##  avg    0.8809
## ============================
```

Train the model by calling the $train() method where tuning parameters can be set inside the function

```
# Train the model
r$train(edx_reco, opts = c(opts$min, nthread = 4, niter = 10))
```

```
## iter      tr_rmse          obj
##    0       0.9537   1.4592e+07
##    1       0.8790   1.3357e+07
##    2       0.8549   1.3071e+07
##    3       0.8437   1.2935e+07
##    4       0.8371   1.2876e+07
##    5       0.8323   1.2831e+07
##    6       0.8290   1.2796e+07
##    7       0.8268   1.2774e+07
##    8       0.8254   1.2763e+07
##    9       0.8243   1.2752e+07
```

```
# Calculate the predicted values (ratings)
y_hat_reco_fin <- r$predict(validation_reco, out_memory())

# Obtain RMSE by comparing to validation set
# ratings
rmse_matfac_fin <- RMSE(validation$rating, y_hat_reco_fin)
print(c("Matrix Factorization RMSE obtained from validation$rating:",
    rmse_matfac_fin))
```

```
## [1] "Matrix Factorization RMSE obtained from validation$rating:"
## [2] "0.834137787726984"
```

```
# Comparing with all the models calculated
kable((rmse_results[1:6, ]), "pandoc", digits = 7,
    caption = "RMSE Results", align = "c")
```

Table 12: RMSE Results

method	RMSE_train_set	RMSE_validation
Base Model - Mean Rating	1.0607079	NA
Movie Effects Model	0.9437144	NA
Movie & User Effects Model	0.8661625	NA
Movie + User + Time Effects Model	0.8660880	NA
Regularized Movie + User Effects Model	0.8655425	NA
Matrix Factorization GD Model	0.8418529	NA

The lowest RMSE from matrix factorization model is relatively close to the RMSE obtained from validation set ratings.

4 Conclusion

4.1 Results and Discussion

In this project, we explore and apply machine learning algorithms in building a recommendation system to predict movie ratings for MovieLens data with 10 million entries. By manipulating the given dataset to create train and test sets from edx dataset, we evaluate linear models with movie, user and time biases. In addition, movie and user effects are regularized but resulting RMSE is above target. Finally, we evaluate matrix factorization model using recosystem package that implements the LIBMF algorithm, and achieve the RMSE which is below target RMSE of 0.8649.

4.2 Limitations and Future Work

The obvious limitation with the current setup within this project is the data refresh mechanism to account for new users, movies and ratings. For the recommender system to be fully functioning in real-time manner, system architecture design needs to account for the assimilation of new information and process design needs to manage the movie recommendation work flow seamlessly.

Matrix factorization yields satisfactory result at the expense of computational time and resources. In order to make this method more scalable, additional servers might be required. On the other hand, optimization of criteria (dim, lrate, nthread) could potentially be investigated to evaluate the impacts on the RMSE value when high-performance computation is realized.

Within the scope of this report, further investigation on genre and its effects on rating predictions with the possibility to reduce RMSE values even further for linear model could be an option.

5 References

1. Rafael A. Irizarry (2019), Introduction to Data Science: Data Analysis and Prediction Algorithms with R
2. Y Koren, R Bell, C Volinsky (2009), Matrix Factorization Techniques for Recommender Systems
3. Yixuan Qiu (2020), recosystem: Recommender System Using Parallel Matrix Factorization https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html

4. Chin, Yuan, et al. (2015), LIBMF: A Matrix-factorization Library for Recommender Systems https://www.csie.ntu.edu.tw/~cjlin/libmf/