Movie Recommendation System

David Oluyole Ajekigbe

13/04/2021

Preface

This report is the first of two study-project requirements needed to complete the last stage (9th of 9 courses) of HarvardX's Data Science Professional Certificate[1] program, 2021.

[1] https://www.edx.org/professional-certificate/harvardx-data-science

Table of Contents

Executive Summary

The Movie Recommendation project is popular and tasky. The Netflix challenge in 2006 made it popular when it put up a million dollars for whoever successfully improves their recommendation system by at least 10%. The data provided by the HarvardX program is smaller and very similar to that used in the Netflix challenge.The recommendation system in this study made use of movieId (movie effect, b_i) and userid (user effect, b_u) to predict movie ratings.

Therfore, the goal for this project was to improve the loss function (RMSE) below a given project goal. Using RMSE to evaluate algorithms gives an idea of how wrong all predictions are (*an RMSE of 0 implies* ***no*** *error, as it tends to 1, the error increases*).

The Regularized linear model and the matrix factorization method were observed to return better RMSE than the project RMSE goal. Most of other models could not work with the computing capacity used(4Gig RAM, 500Gig HDD). However, the both models which returned better models tooK several hours to compute due to the very large dataset.

The study ended with conclusions, suggestions, recommendation for future work and study limitation.

CHAPTER 1

INTRODUCTION

Movie Recommendation System

In this jet age of fast food, fast internet, fast machines and robots, it has become very important to live up to so many expectations in a short period of time. At work, home and school you have to be multitasking and make accurate decisions in the shortest time. Humans have hence depended on machines/robots/algorithms to help them make such decisions.

This has made recommendation systems very important and available in every sphere of our life, to make ease our decison making process and help us to make the right choices. The recommendation system aims to assist us by suggesting things that should be appealing to us given the data it has been able to gather directly or indirectly.

The artificial intelligence makes use of data such as browsing history, previous purchase pattern, location, device type, demographic and social factors and so on to arrive at meaningful conclusions to what the user might need. The huge dependency of human activities on artificial intelligence such as recommendation systems makes it absolutely important for it to be as accurate as possible.

The movie recommendation system that is built in this project made use of features such as 'userid' and 'movieid' to predict the rating that would be given to the movie by the user. In real life, there are usually more feature variables available to train the model. Other similar artificial intelligence applications close to the movie recommendation system is the E-Commerce where businesses use it to sell, cross-sell and announce new products to potential customers. Another example is the social media friends and story recommendations. They use demographic and interest data to bring you stories that interest you, connect you with people you should/may/want to know all with the purpose of a sweet experience on their platforms.

The movies recommendation model has been built by several people with different approaches. However, although recommendation algorithms are time and Computer memory consuming, attempt is made in this project to develop the most appropriate model for the Movielens data given the time and resources at my disposal.

Description of MovieLens Dataset

The particular dataset used by Netflix in the Netflix 2006 challenge has not been made available till date. However, the GroupLens research lab2 generated their own database with over 20 million ratings for over 27,000 movies by more than 138,000 users. The data used in this project is a subset of the GroupLens research lab (about 10million entries). The codes to mine and clean the dataset were provided by the HarvardX Datascience program.

Study Goals

The following are the goals of this study:

1. To predict movie rating given certain features.

2. To use at least two models for the prediction.

3. To select the best model using their RMSE values after testing the validation set on the EDX set.

Steps Performed

To arrive at the desired goal, the following steps were performed:

1. The data used was downloaded from the given webpage, it was scrapped, and cleeaned. The dataset was partitioned to EDX and validation set using 9:1 ratio respectively. The EDX set was further devided into train.set and test.set in the 8:2 retio respectively and made ready for analysis.

2.The exploratory data analysis was carried out on the edx and train partitions with vixualizations and descriptive estimates for inspection.

3. The data was further shreaded for unnecessary variables which comprised or the 'timestamp' and 'genre' variables for reasons such as to allow for enough computing power.

4. Different models and parameters were developed from random prediction to regularization and to matrix factorization process.The model with the least RMSE lost value is picked for validation.

5. When the desireable model has been tested, it is further validated by testing the validation dataset on the edx dataset for consistency.

CHAPTER TWO

EXPLORATORY DATA ANALYSIS OF MOVIELENS DATA

The codes below were given by the management of EDX Harvard Data Science with R for a unified data mining process and outcome. Running these codes took some time to generate and clean the data.

The validation set comprised of 10% of the movielens data while the rest is called the edx set. 80% of edx set was assigned to the train test while the test set had 20% of the edx set. Training set is given so much because the pattern identified by the model is extracted from it, therefore the opinion that as much as possible data should be given to the trainset data to build the model. The small fractions for test set and validation set each is used because this data is large, a 10% hence is a large number, enough for testing and validation set.

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

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

## Loading required package: tidyverse

## -- Attaching packages --------------------------------------- tidyverse
1.3.1 --

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.0     v dplyr   1.0.5
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## Warning: package 'ggplot2' was built under R version 4.0.5

## Warning: package 'tibble' was built under R version 4.0.5

## Warning: package 'tidyr' was built under R version 4.0.5

## Warning: package 'readr' was built under R version 4.0.5

## Warning: package 'dplyr' was built under R version 4.0.5

## Warning: package 'forcats' was built under R version 4.0.5

## -- 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
```

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

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

## Warning: package 'data.table' was built under R version 4.0.5

##
## 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
library(tidyverse)
library(caret)
library(data.table)
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 4.0.5

##
## 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
library(ggthemes)
library(scales)
```

```
##
## Attaching package: 'scales'
## The following object is masked from 'package:purrr':
## 
##     discard

## The following object is masked from 'package:readr':
## 
##     col_factor

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

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



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

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")),
"\\::", 3)
  colnames(movies) <- c("movieId", "title", "genres")

# if using R 3.6 or earlier:
#movies <- as.data.frame(movies) %>% mutate(movieId =
as.numeric(levels(movieId))[movieId],
                                           #title = as.character(title),
                                           #genres = as.character(genres))
# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))


movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use
`set.seed(1)`

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

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

```r
# 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)

#Here, the edx which is 90% of the MovieLens Data has 9000047 observations
and 6 variables.
#the edx set will further be partitioned into train and test sets
#the validation set which is 10% of the movielens data has 999999
observations and 6 variables

############## EXTRACT TRAIN AND TEST SETS FROM EDX SET
#########################
#We will be using 80% of the edx set as the train.set and 20% as test set

set.seed(1, sample.kind="Rounding")

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

testindex <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list =
FALSE)
train.set <- edx[-testindex,]
temp1 <- edx[testindex,]

#The partitioned train and test sets from the edx dataset contains 7200037
and 1800010 observations respectively.
#they both have 6 variables each

# ensure that userId and movieId in test.set are also in train.set
test.set <- temp1 %>%
  semi_join(train.set, by = "movieId") %>%
  semi_join(train.set, by = "userId")

# Add rows removed from test.set set back into train set
removed_obs <- anti_join(temp1, test.set)
```

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

train.set <- rbind(train.set, removed_obs)

rm(temp1, testindex, removed_obs)
```

Exploratory Data ANalysis

In this section, each of the variables in the train.set datasets (that is, the dataset we want to train) is presented. Aside understanding our dataset, data exploration gives a rough insight on what to expect from the model(s). We can also get information on variables to include or exclude and which model to use knowing the dependent and independent variables. In this model building, the movie title is our dependent variable.

Exploring the train.set dataset

```
names(train.set)

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

dim(train.set)

## [1] 7200089       6
```

There are 6 variables in the data.set. Each of them are listed in the output above.

```
str(train.set)

## Classes 'data.table' and 'data.frame':   7200089 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  185 316 329 355 364 377 420 539 588 589 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838983525 838983392 838983392 838984474 838983707
838983834 838983834 838984068 838983339 838983778 ...
##  $ title    : chr  "Net, The (1995)" "Stargate (1994)" "Star Trek:
Generations (1994)" "Flintstones, The (1994)" ...
##  $ genres   : chr  "Action|Crime|Thriller" "Action|Adventure|Sci-Fi"
"Action|Adventure|Drama|Sci-Fi" "Children|Comedy|Fantasy" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

The userid and timestamp are of the integer class, movieid and rating are numeric while title and genres are characters. From this output, an abnormality can be seen from the timestamp being integer. Looking at the sample data from the timestamp variable, the entries are given in numbers which needs to be converted to the proper timesamp format of date and time.

```
train.set$timestamp <- as.Date(as.POSIXct(train.set$timestamp, origin =
"1970-01-01" ))
class(train.set$timestamp)

## [1] "Date"
```

Now, the class of the timestamp variable has been changed to date.

From the result, movieId is numeric whereas, upon inspection, movieId is integer. Hence, converted movieId from numeric to integer.

```
train.set$movieId <- as.integer(train.set$movieId)
class(train.set$movieId)

## [1] "integer"
```

Now that our movieid variable is an integer, we can go ahead with the Exploratory Data Analysis.

For a more robust breakdown of the train.set data, the summary command is used.

```
summary(train.set) #These summarizes all the variables in the dataset

##      userId          movieId          rating        timestamp         
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :1995-01-09  
##  1st Qu.:18127   1st Qu.:  648   1st Qu.:3.000   1st Qu.:2000-01-01  
##  Median :35750   Median : 1834   Median :4.000   Median :2002-10-24  
##  Mean   :35873   Mean   : 4122   Mean   :3.512   Mean   :2002-09-20  
##  3rd Qu.:53611   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:2005-09-14  
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :2009-01-05  
##     title              genres         
##  Length:7200089     Length:7200089    
##  Class :character   Class :character  
##  Mode  :character   Mode  :character  
## 
## 
## 
```

It can be observed that the minimum rating is 1 and the maximum is 5.0. The earliest date is 1995-01-09 while the recent date is 2009-01-05. The Title variable is a character type, hence cannot be used in the model building.

Basic Exploration

Some basic descriptions of the train.set data are given below.

```
#Since the mean of ratings is about 3 (from the summary output), we can check
number of ratings greater than 3
#Numbers of ratings greater than 3
sum(train.set$rating > 3)

## [1] 4237542

#Numbers movies available for rating
length(unique(train.set$movieId))

## [1] 10677
```

```
#Checking Genres Variable:
#Numbers of Genres available without repetition
length(unique(train.set$genres))

## [1] 797

#Number of users (userid) without repetition
length(unique(train.set$userId))

## [1] 69878
```

Exploring the data by Genres

From the above OUTPUT of number of genres, there are 797 genres categories. To check the frequency of each genre in the train.set dataset, the code below is executed.

```
train.set %>% group_by(genres) %>%
  summarise(Freq=n()) %>%
  head()

## # A tibble: 6 x 2
##   genres                                             Freq
##   <chr>                                             <int>
## 1 (no genres listed)                                    5
## 2 Action                                            19576
## 3 Action|Adventure                                  54716
## 4 Action|Adventure|Animation|Children|Comedy         5946
## 5 Action|Adventure|Animation|Children|Comedy|Fantasy   148
## 6 Action|Adventure|Animation|Children|Comedy|IMAX       50
```

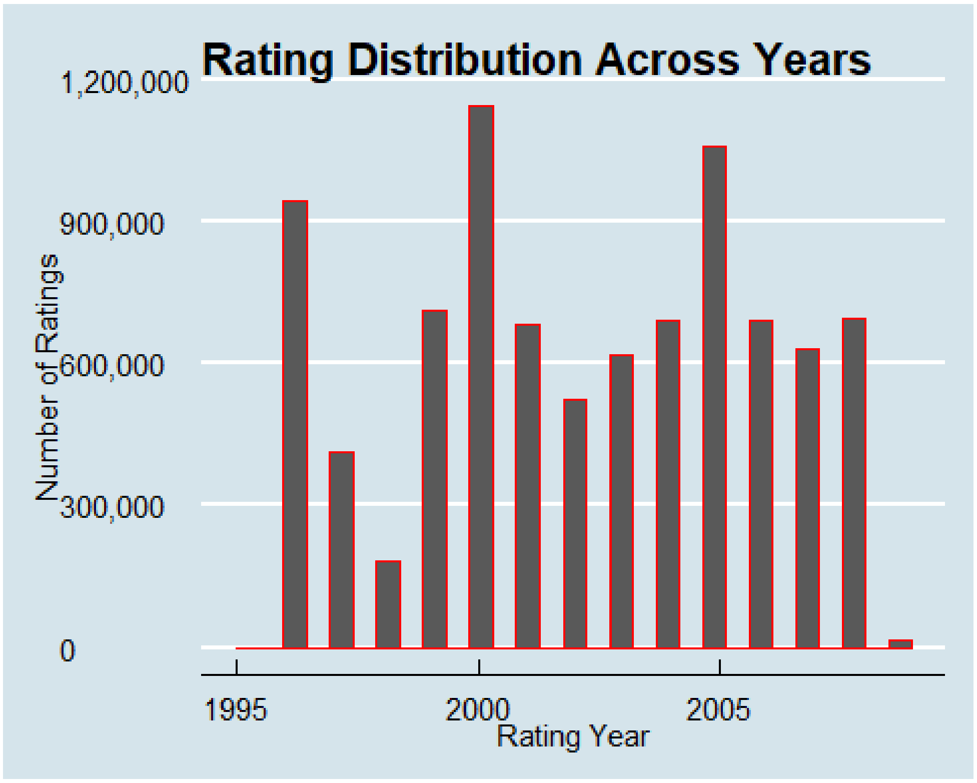
Exploring Throught The Date variable

From the summary output above, the minimum from the time stamp summary (which is also the date) was 1995-01-09 while the maximum (that is, the end date) was on 2009-01-05. The interval, that is, how long the rating exercise lasted can be arrived at as:

```
interval_in_days = difftime(max(train.set$timestamp),
min(train.set$timestamp), units = "days")
interval_in_years = as.double(interval_in_days)/365 # absolute years
interval_in_years

## [1] 14
```

Movie Rating Distribution Across years

```
edx %>% mutate(rating_year = year(as_datetime(timestamp, origin="1970-01-
01"))) %>%
  ggplot(aes(x=rating_year)) +
  geom_histogram(color = "red") +
  ggtitle("Rating Distribution Across Years") +
  xlab("Rating Year") +
  ylab("Number of Ratings") +
  scale_y_continuous(labels = comma) +
  theme_economist()

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



From the graph above, more people participated in the rating exercise in year 2000. The year 1998 had the least participation in the movies rating exercise.

Exploring the ratings from the EDX Dataset

Without using any model, we can use frequencies to explore the top 10 of the movies with highest number of rating.

```
numRatings <- edx %>% group_by(movieId) %>%
  summarize(numRatings = n(), movieTitle = first(title)) %>%
  arrange(desc(numRatings)) %>%
  top_n(10, numRatings)
numRatings

## # A tibble: 10 x 3
##    movieId numRatings movieTitle
##      <dbl>      <int> <chr>
##  1     296      31362 Pulp Fiction (1994)
##  2     356      31079 Forrest Gump (1994)
##  3     593      30382 Silence of the Lambs, The (1991)
##  4     480      29360 Jurassic Park (1993)
##  5     318      28015 Shawshank Redemption, The (1994)
##  6     110      26212 Braveheart (1995)
##  7     457      25998 Fugitive, The (1993)
##  8     589      25984 Terminator 2: Judgment Day (1991)
##  9     260      25672 Star Wars: Episode IV - A New Hope (a.k.a. Star
Wars) (19~
## 10     150      24284 Apollo 13 (1995)
```

Exploring Dates by Ratings

The frequencies of movie ratings per date is given with the following codes:

```
edx %>% mutate(date_rating = date(as_datetime(timestamp, origin="1970-01-
01"))) %>%
  group_by(date_rating, title) %>%
  summarise(freq = n()) %>%
  arrange(-freq) %>%
  head(5)

## `summarise()` has grouped output by 'date_rating'. You can override using
the `.groups` argument.

## # A tibble: 5 x 3
## # Groups:   date_rating [3]
##   date_rating title
freq
##   <date>      <chr>
<int>
## 1 1998-05-22  Chasing Amy (1997)
322
## 2 2000-11-20  American Beauty (1999)
277
## 3 1999-12-11  Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)
254
## 4 1999-12-11  Star Wars: Episode V - The Empire Strikes Back (1980)
251
## 5 1999-12-11  Star Wars: Episode VI - Return of the Jedi (1983)
241
```

Rank the 10 levels of different ratings

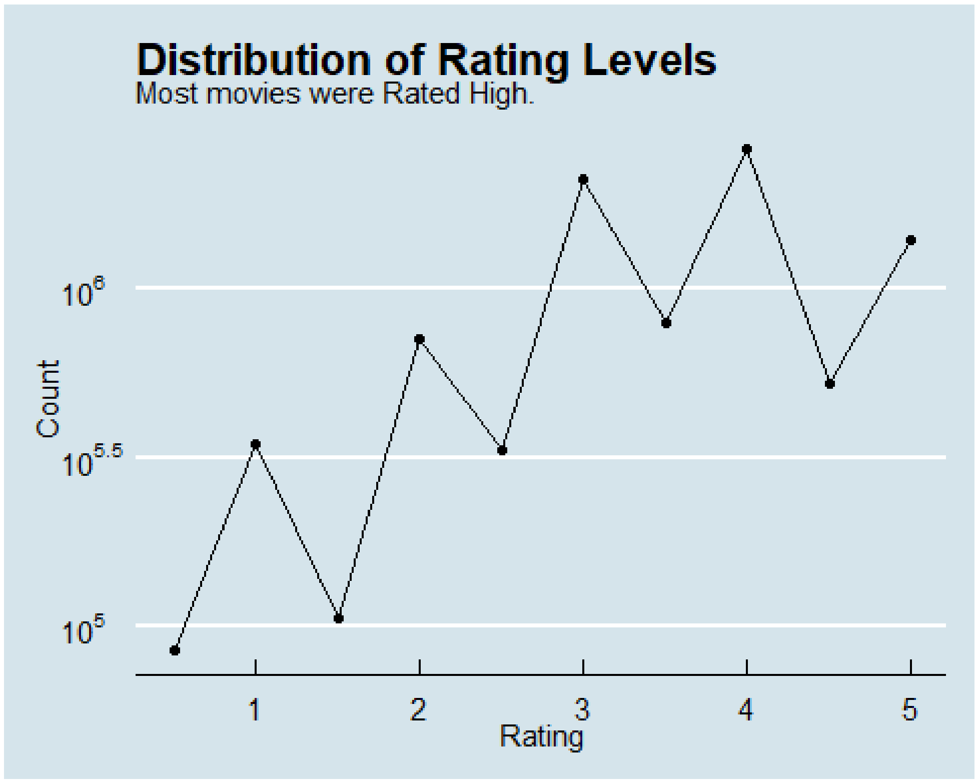
The possible ratings are from 0.5 to 5.0 with an interval of 0.5. In this section, we are displaying number of times each rating was made used of.

```
rankRatings <- edx %>% group_by(rating) %>%
  summarize(freq = n())
rankRatings

## # A tibble: 10 x 2
##    rating    freq
##     <dbl>   <int>
##   1    0.5   85374
##   2    1    345679
##   3    1.5  106426
##   4    2    711422
##   5    2.5  333010
##   6    3   2121240
##   7    3.5  791624
##   8    4   2588430
##   9    4.5  526736
## 10    5   1390114
```

The rating 4.0 was most given to movies followed by rating 3.0 with frequencies of 2,588,430 and 2,121,422 respectively. The level 4 rating was assigned the most for the movies, followed by 3, 5, 3.5 and lastly 2 was the least of the top 5

The output above can be presented in a graphical format:

```
edx %>%
  group_by(rating) %>%
  summarize(count = n()) %>%
  ggplot(aes(x = rating, y = count)) +
  geom_line() +
  geom_point() +
  scale_y_log10(breaks = trans_breaks("log10", function(x) 10^x),
                labels= trans_format("log10", math_format(10^.x)))+
  ggtitle("Distribution of Rating Levels", subtitle = "Most movies were Rated
High.")+
  xlab("Rating")+
  ylab("Count")+
  theme_economist()
```



Exploring the movies variable

It has been earlier established that there are 10677 movies in the EDX dataset. The ratings range from 0.5 to 5 with an interval of 0.5.

The number of movies is given as:

The number of unique movies, without repeatition is given as:

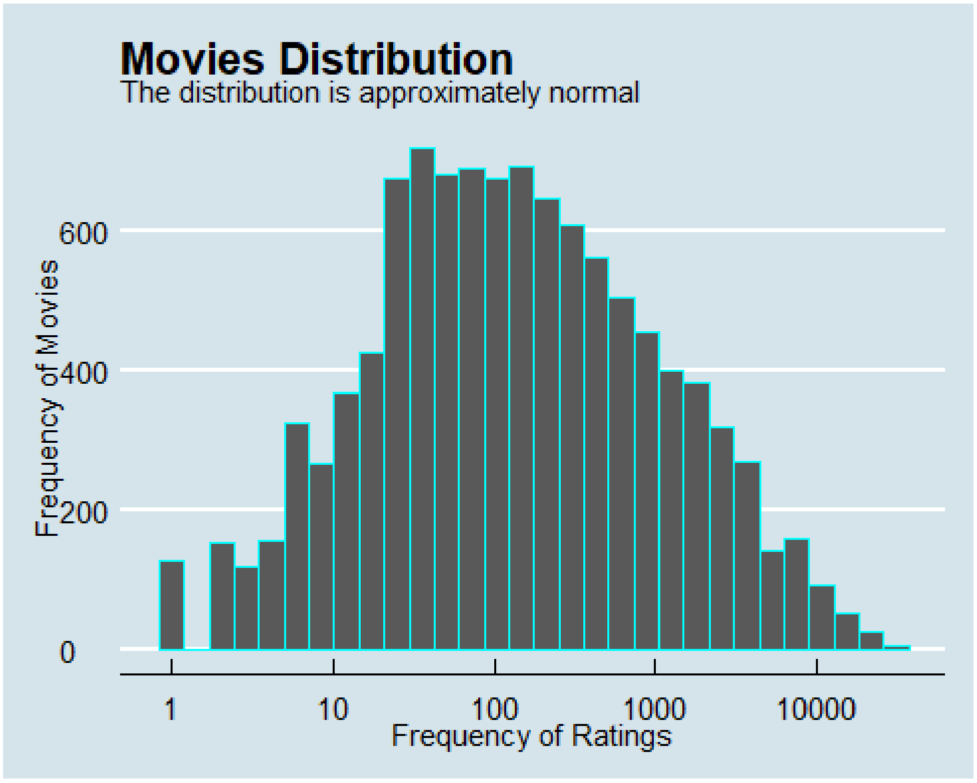
```
length(unique(edx$movieId))

## [1] 10677
```

Distribution of movies

To check if the movies ratings follow a normal distribution using the histogram:

```
edx %>% group_by(movieId) %>%
  summarise(n=n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bin = 30, color = "cyan") +
  scale_x_log10() +
  ggtitle("Movies Distribution",
          subtitle = "The distribution is approximately normal") +
  xlab("Frequency of Ratings")+
  ylab("Frequency of Movies")+
  theme_economist()
```

```
## Warning: Ignoring unknown parameters: bin

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



Exploring the 'userid' Variable in the EDX set

The users variable uniquely identifies those rating the movies.

```
length(unique(edx$userId))

## [1] 69878
```

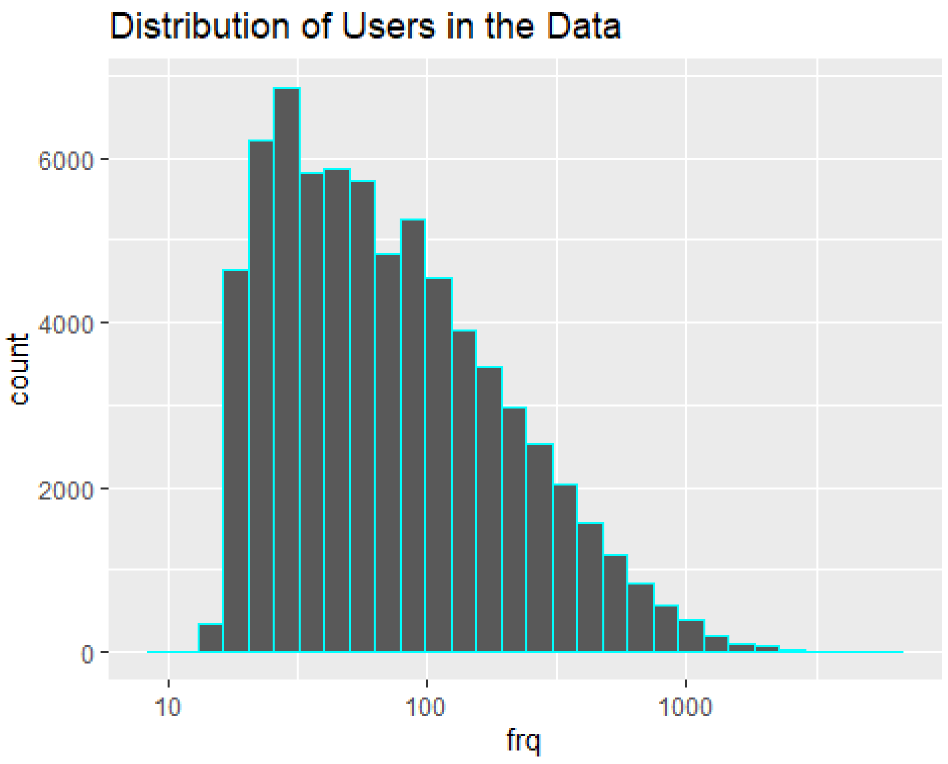
There are 69878 distinct users in the edx dataset. To check the distribution of the userid variable, observing the first six for the display of userid and how many times they appear in the data.

```
edx %>% group_by(userId) %>%
  summarise(freq=n()) %>%
  arrange(freq) %>%
  head()

## # A tibble: 6 x 2
##   userId  freq
##    <int> <int>
## 1  62516    10
## 2  22170    12
## 3  15719    13
## 4  50608    13
## 5    901    14
## 6   1833    14
```

Graphical Inspection of users that have rated movies

```r
edx %>% group_by(userId) %>%
  summarise(frq=n()) %>%
  ggplot(aes(frq)) +
  geom_histogram(bin = 30, color = "cyan") +
  scale_x_log10() +
  ggtitle("Distribution of Users in the Data")
```

```
## Warning: Ignoring unknown parameters: bin

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



Data Preparation and Methodology

The objective of this analysis is to predict user rating with the feature variables. This work, due to lack of sufficient computer capacity, efficiency and speed, I used the userid and movieid to predict the rating of movies. Therefore I subset only the needed variables and let go off the timestamp and genre.

```r
train.set <- train.set %>% select(userId, movieId, rating, title, genres)
test.set <- test.set %>% select(userId, movieId, rating, title, genres)
validation <- validation%>% select(userId, movieId, rating, title, genres)
```

The genre variable is a character type, this will not work with our model fitting until converted to factor. To convert character to factor, the codes below were used:

```
train.set$genres<- as.factor(train.set$genres)
test.set$genres<- as.factor(test.set$genres)
validation$genres<- as.factor(validation$genres)
```

Now, to confirm that the genres variables in the three datasets have been changed from character to factor:

```
str(train.set$genres)

##  Factor w/ 797 levels "(no genres listed)",..: 187 98 71 460 274 241 128
539 267 244 ...

str(test.set$genres)

##  Factor w/ 787 levels "(no genres listed)",..: 574 210 539 307 120 152 486
407 98 249 ...

str(validation$genres)

##  Factor w/ 773 levels "Action","Action|Adventure",..: 474 96 437 202 575
86 361 686 442 330 ...
```

The train set having greater levels (797) than the test set (787) and validation set (773) is a good indication that all the possible genres in test set and validation set were included or accounted for in the train set.

We are building models to predict the movie 'ratings' using movieid (numeric) and title which is character (both are categorical variables) while the ratings is a numeric variable. Although the dependent varable (rating) is numeric, it has only 10 possible outputs hence this qualifies it as a categorical variable as well. The independent/predictor variables are chategorical variables and the dependent/predicted variable is also a categorical variable. Therefore, the problem at hand is a classification problem. Therefore, some of the appropriate models that can work with a commodity laptop and this volume of data are:

1. Random Prediction

2. Mean

3. Partial Linear Model (Movie effect)

4. Linear Model (Full) (Movie effect and user effect)

5. Regularized Linear Model

6. Matrix Factorization using Recosystem

The simple model using the mean () for prediction is given as:

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

with $\varepsilon_{i,u}$ independent errors sampled from the same distribution centered at 0 and μ the "true" rating for all movies. We know that the estimate that minimizes the RMSE is the least squares estimate of μ and, in this case, is the average of all ratings.

The study will then flow to adding the movie effect or movie bias to the model above as:

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

We refer to the b_is as movie effects or bias. This model is estimated using the least square (lm) function.

A further improvement to our model will be to account for the user effect as stated in the model below:

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

where b_u is a user-specific effect or bias. Thesse improvements are meant to keep improving the RMSE estimates of the model.

However, when making predictions, we need one number, one prediction, not an interval. For this, we introduce the concept of regularization.

Regularization

Regularization is the class of methods needed to modify maximumlikelihood in machine learning to give reasonable answers in unstable situations, (Bickel & Li, 2006). In addition to being able to evaluate our function, we would like a solution fromapplying our learning algorithm to noisy data to be "near" the optimal solution withoutnoise. There are two parts to measuring the success of our algorithm given sometraining data. First, we would like to match the training data as closely as possible.However, keeping in mind that there may be noise in the training data, we would liketo prevent overfitting by restricting the complexity of the functions we are studying.Regularization aims to balance these two conflicting requirements, (Ong, 2005).

Regularization permits us to penalize large estimates that are formed using small sample sizes. It has commonalities with the Bayesian approach that shrunk predictions. The general idea behind regularization is to constrain the total variability of the effect sizes. Why does this help? Consider a case in which we have movie $i = 1$ with 100 user ratings and 4 movies $i = 2,3,4,5$ with just one user rating. We intend to fit the model.

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

Matrix factorization

Matrix Factorization is a technique to discover the latent factors from the ratings matrix and to map the movies and the users against those factors. The approach is similar to expressing a number, 61, as a product of two numbers. As a prime number, it is not possible to decompose 61 as a product of two numbers, but the original number can be closely approximated with 6×10. The error in this decomposition is 1, (Vijay & Deshpande, 2019).

Many complex matrix operations cannot be solved efficiently or with stability using the limited precision of computers. Matrix decompositions are methods that reduce a matrix into constituent parts that make it easier to calculate more complex matrix operations. Matrix decomposition methods, also called matrix factorization methods, are a foundation of linear algebra in computers, even for basic operations such as solving systems of linear equations, calculating the inverse, and calculating the determinant of a matrix, (Jason Brownlee, 2019).

Matrix Factorizationis a widely used concept in machine learning. It is very much related to factor analysis, singular value decomposition (SVD), and principal component analysis (PCA). Here we describe the concept in the context of movie recommendation systems. (Rafael A. Irizarry- Introduction to Data Science)

We have described how the model:

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

accounts for movie to movie differences through the b_i and user to user differences through the b_u. But this model leaves out an important source of variation related to the fact that groups of movies have similar rating patterns and groups of users have similar rating patterns as well. We will discover these patterns by studying the residuals:

$$r_{u,i} = y_{u,i} - \hat{b}_i - \hat{b}_u$$

To see this, we will convert the data into a matrix so that each user gets a row, each movie gets a column, and $y_{u,i}$ is the entry in row u and column i

CHAPTER 3

MODEL FITTING AND RESULTS

The Netflix movie recommendation challenge made use of the RMSE in accessing the accuracy of the model. RMSE is the default metrics used to evaluate algorithms on regression datasets in caret. RMSE or Root Mean Squared Error is the average deviation of the predictions from the observations. It is useful to get a gross idea of how well (or not) an algorithm is doing, in the units of the output variable, (Jason Brownlee, 22020).

The RMSE is systematically derived through the Mean Absolute Error (MAE) and Mean Squared Error (MSE) below:

Define Mean Absolute Error (MAE)

The mathematical expression for the loss function of the Root Mean Square Error (RMSE) given below is derived systematically by defining the MAE - Mean Absolute Error in R:

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |\, \hat{y}_i - y_i|$$

The definition as coded into R is givrn as:

```
MAE <- function(true_ratings, predicted_ratings){
  mean(abs(true_ratings - predicted_ratings))
}
```

Define Mean Squared Error (MSE)

The square of the MAE gives the Mean Square Error which is mathematically expressed as:

$$MSE = \frac{1}{N}\sum_{u,i} (\, \hat{y}_i - y_i)^2$$

The definition as coded into R is given as:

```
MSE <- function(true_ratings, predicted_ratings){
  mean((true_ratings - predicted_ratings)^2)
}
```

Define Root Mean Squared Error (RMSE)

The Root Mean Square Error is finnaly derived by taking the root of the Mean Square Error is mathematically expressed below as:

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

In the three formulas above, N is the defined as the frequency of ratings, $y_{u,i}$ stands for the rating of movie i by user u and $\hat{y}_{u,i}$ represents the prediction of movie i by user u

To put the RMSE in the R environment, we use:

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Samller errors yield smaller MAE, MSE and RMSE and larger error yields otherwise proportionately.

#Model1 - Linear Model

Runing the *lm* function was not successful as the computer used does not have the computing capacity and storage to perform such large operation.

Therefore, to get around this and fix the Linear prediction model, each of the model parameters will be built individually: the *random prediction*, the *user bias*, the *movie bias* and finally all put together to form the linear model.

```
set.seed(1952, sample.kind = "Rounding")

## Warning in set.seed(1952, sample.kind = "Rounding"): non-uniform
'Rounding'
## sampler used

# Create the probability of each rating
p <- function(x, y) mean(y == x)
rating <- seq(0.5,5,0.5)
# Estimate the probability of each rating with Monte Carlo simulation
B <- 10^3
M <- replicate(B, {
  s <- sample(train.set$rating, 100, replace = TRUE)
  sapply(rating, p, y= s)
})
prob <- sapply(1:nrow(M), function(x) mean(M[x,]))
# Predict random ratings
y_hat_random <- sample(rating, size = nrow(test.set),
                       replace = TRUE, prob = prob)
# Create a table with the error results
result <- tibble(Method = "Project Goal", RMSE = 0.8649, MSE = NA, MAE = NA)
result <- bind_rows(result,
                    tibble(Method = "Random prediction",
                           RMSE = RMSE(test.set$rating, y_hat_random),
                           MSE = MSE(test.set$rating, y_hat_random),
                           MAE = MAE(test.set$rating, y_hat_random)))
# Show the RMSE improvement
result %>% knitr::kable(caption = "RMSE Display of Models")
```

RMSE Display of Models

Method	RMSE	MSE	MAE
Project Goal	0.864900	NA	NA
Random prediction	1.500706	2.252118	1.166805

While the project goal is an RMSE of 0.864900, the random prediction has shown not to be sufficient for this data to predict a trustworthy output. The random prediction has an RMSE of 1.5007061, this is a problem of over fitting.

The mean rating will also be used to model this data, if it would give a bettwe RMSE. The mean rating, that is, the slope of the linear model

```r
# The initial prediction is the mean of the ratings (mu).
# y_hat = mu
# Mean of observed values
mu <- mean(train.set$rating)
# Update the error table
result <- bind_rows(result,
                    tibble(Method = "Mean",
                           RMSE = RMSE(test.set$rating, mu),
                           MSE  = MSE(test.set$rating, mu),
                           MAE  = MAE(test.set$rating, mu)))
# Show the RMSE improvement
result %>% knitr::kable(caption = "RMSE Display of Models")
```

RMSE Display of Models

Method	RMSE	MSE	MAE
Project Goal	0.864900	NA	NA
Random prediction	1.500706	2.252118	1.1668051
Mean	1.059904	1.123397	0.8552175

Although using the mean prediction model gives a better RMSE (1.059904) value than the random prediction (1.502071), it still has the overfitting attribute which is not acceptable as a good prediction accuracy. The mean prediction can be improved towards forming the linear model. This mean prediction has not taken into consideration the effect of the movieid nor the userid. Accounting for both parameters in the linear model should result in a better RSME estimate.

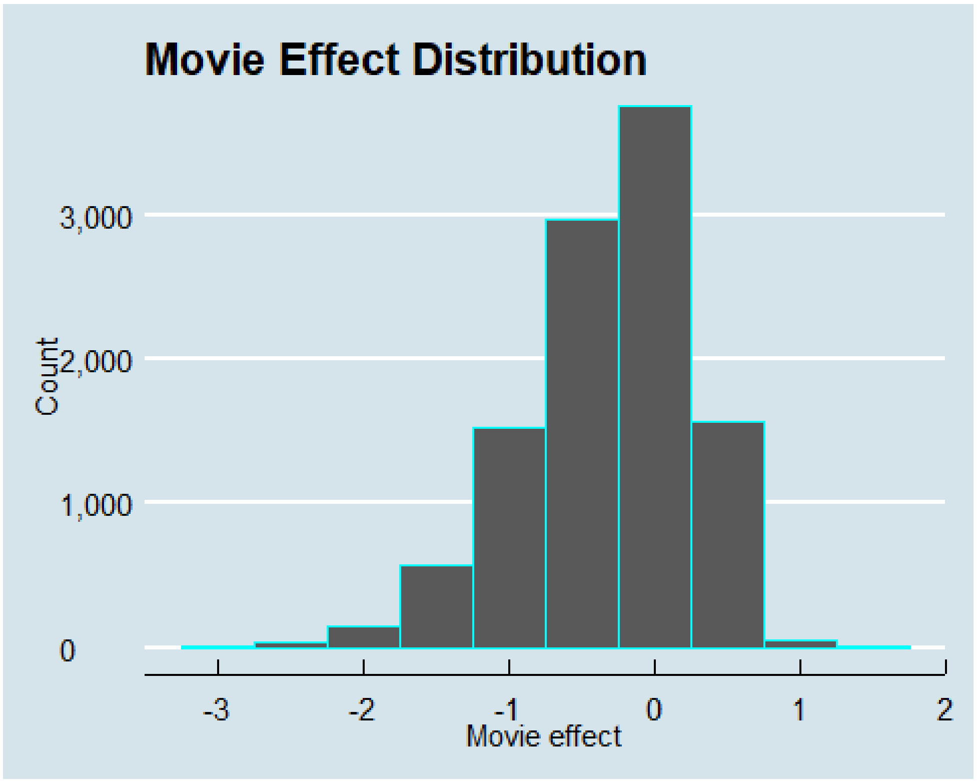
Include movie effect (b_i)

This partial model accounts for only the movie effect:

$$Y_i = \mu + b_i + \varepsilon_i$$

```r
# bi is the movie effect (bias) for movie i.
# y_hat = mu + bi
# Movie effects (bi)
```

```
bi <- train.set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
head(bi)

## # A tibble: 6 x 2
##   movieId    b_i
##     <int>  <dbl>
## 1       1  0.418
## 2       2 -0.308
## 3       3 -0.371
## 4       4 -0.645
## 5       5 -0.448
## 6       6  0.303

# Plot the distribution of movie effects
bi %>% ggplot(aes(x = b_i)) +
  geom_histogram(bins=10, col = I("cyan")) +
  ggtitle("Movie Effect Distribution") +
  xlab("Movie effect") +
  ylab("Count") +
  scale_y_continuous(labels = comma) +
  theme_economist()
```



```
# Predict the rating with mean + bi
y_hat_bi <- mu + test.set %>%
  left_join(bi, by = "movieId") %>%
```

```r
  .$b_i
# Calculate the RMSE
result <- bind_rows(result,
                    tibble(Method = "Mean + bi",
                           RMSE = RMSE(test.set$rating, y_hat_bi),
                           MSE = MSE(test.set$rating, y_hat_bi),
                           MAE = MAE(test.set$rating, y_hat_bi)))

# Show the RMSE improvement
result %>% knitr::kable(caption = "RMSE Display of Models")
```

RMSE Display of Models

Method	RMSE	MSE	MAE
Project Goal	0.8649000	NA	NA
Random prediction	1.5007057	2.2521177	1.1668051
Mean	1.0599043	1.1233971	0.8552175
Mean + bi	0.9437429	0.8906507	0.7381140

Adding the movie effect further reduces or improves the RMSE towards the project RMSE goal. Accounting for the movie effect, the RMSE of the linear model becomes 0.9437429 which is still greater than the project goal RMSE of 0.864900.

Linear Model (movie & user bias)

To further improve the linear model, the user effect (bu) is accounted for. Our model becomes:

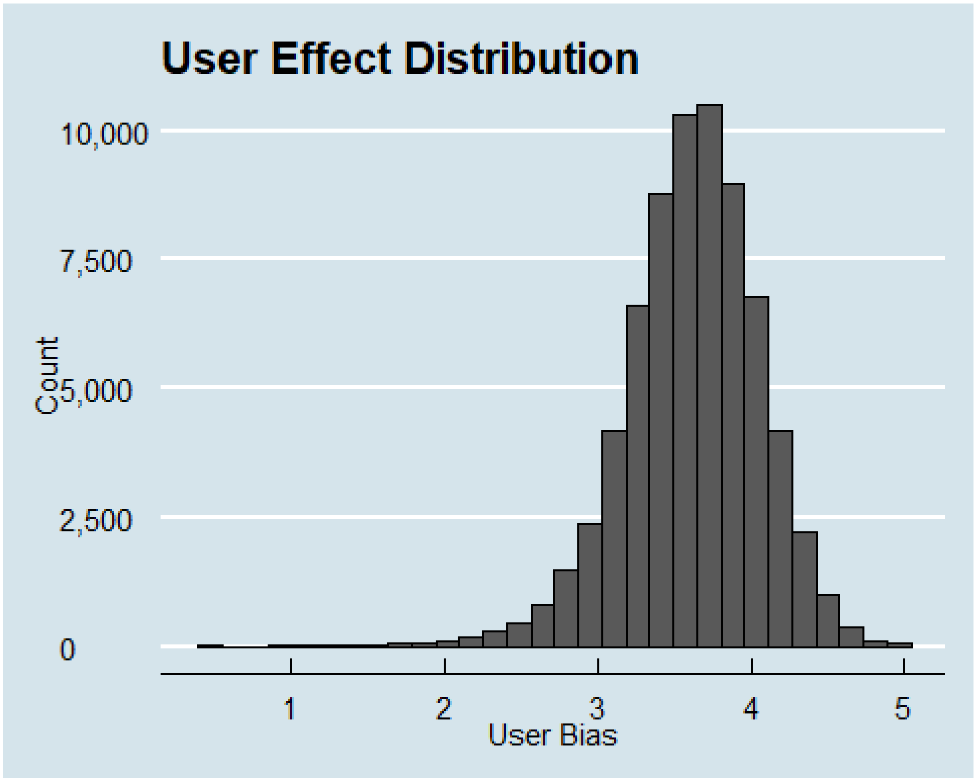
$$Y_{u,i} = \mu + b_i + b_u + \varepsilon_{u,i}$$

where User effect is

$$bu$$

```r
bu <- train.set %>%
  left_join(bi, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))
# Prediction
y_hat_bi_bu <- test.set %>%
  left_join(bi, by='movieId') %>%
  left_join(bu, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

# Plot the distribution of user effects
train.set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(color = "black") +
  ggtitle("User Effect Distribution") +
  xlab("User Bias") +
  ylab("Count") +
  scale_y_continuous(labels = comma) +
  theme_economist()
```

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



```
# Update the results table
result <- bind_rows(result,
                    tibble(Method = "Mean + bi + bu",
                           RMSE = RMSE(test.set$rating, y_hat_bi_bu),
                           MSE  = MSE(test.set$rating, y_hat_bi_bu),
                           MAE  = MAE(test.set$rating, y_hat_bi_bu)))
# Show the RMSE improvement
result %>% knitr::kable()
```

Method	RMSE	MSE	MAE
Project Goal	0.8649000	NA	NA
Random prediction	1.5007057	2.2521177	1.1668051
Mean	1.0599043	1.1233971	0.8552175
Mean + bi	0.9437429	0.8906507	0.7381140

Mean + bi + bu	0.8659319	0.7498381	0.6694465

The addition of both user effect and movie effect further improved the RMSE (0.8659319) to a great extent when compared with the project RMSE goal of (0.8649000). The linear model is hence completed with a full model fitting approximately close to the project goal.

It can be observed that the addition of movie effect and user further improved the RMSE towards the project RMSE goal. Accounting for the genre effect, the RMSE of the linear model should hopefully become more improved.

Model 3 - Regularization

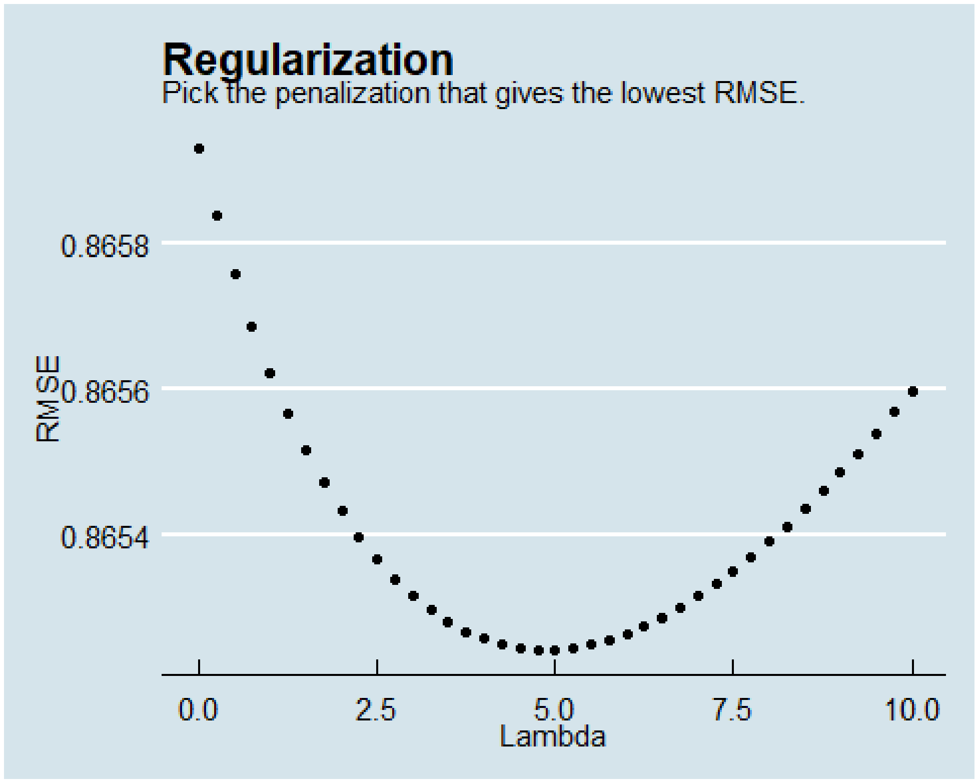
Regularization is an improvement of the linear model. Although the linear model provided a good and approximately close estimation for the ratings relative to the project RMSE goal, the Regularization method claims to provide an improvement to the prediction if the the movies are penalized with few number of ratings. This is done by adding a value, usually referred to as lambda, to the number of ratings. The act of adding lambda to ratings is called regularization.

However, it is important to estimate the right quantity of lambda, small values of lambda have large effect on small sample sizes and almost no impact for movies with many ratings, while large lambdas can drastically reduce the impact of movies with few ratings.

Therefore it is important to find the lambda that provides the optimal prediction, i.e. that gives us the most preferred and the lowest RMSE.

When the movie and user effects were regularized, the RMSE estimate is indeed further improved to 0.86500 which is closer to the Project RMSE goal, 0.8649000.

```r
regularization <- function(lambda, trainset, testset){

  # Mean
  mu <- mean(trainset$rating)

  # Movie effect (bi)
  b_i <- trainset %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+lambda))

  # User effect (bu)
  b_u <- trainset %>%
    left_join(b_i, by="movieId") %>%
    filter(!is.na(b_i)) %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

  # Prediction: mu + bi + bu
  predicted_ratings <- testset %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    filter(!is.na(b_i), !is.na(b_u)) %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, testset$rating))
}

# Define a set of lambdas to tune
lambdas <- seq(0, 10, 0.25)

# Update RMSES table
rmses <- sapply(lambdas,
                regularization,
                trainset = train.set,
                testset = test.set)

# Plot the lambda x RMSE
tibble(Lambda = lambdas, RMSE = rmses) %>%
  ggplot(aes(x = Lambda, y = RMSE)) +
    geom_point() +
    ggtitle("Regularization",
            subtitle = "Pick the penalization that gives the lowest RMSE.") +
    theme_economist()
```



Now, the lambda that gives the smallest RMSE is applied to the linear model to 'regularize' it:

```
# We pick the lambda that returns the lowest RMSE.
lambda <- lambdas[which.min(rmses)]

# Then, we calculate the predicted rating using the best parameters
# achieved from regularization.
mu <- mean(train.set$rating)

# Movie effect (bi)
b_i <- train.set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

# User effect (bu)
b_u <- train.set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

# Prediction
y_hat_reg <- test.set %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

# Update the result table
result <- bind_rows(result,
                    tibble(Method = "Regularized bi and bu",
                           RMSE = RMSE(test.set$rating, y_hat_reg),
                           MSE  = MSE(test.set$rating, y_hat_reg),
                           MAE  = MAE(test.set$rating, y_hat_reg)))

# Regularization made a small improvement in RMSE.
result

## # A tibble: 6 x 4
##   Method                 RMSE    MSE    MAE
##   <chr>                 <dbl>  <dbl>  <dbl>
## 1 Project Goal          0.865 NA     NA
## 2 Random prediction     1.50   2.25   1.17
## 3 Mean                  1.06   1.12   0.855
## 4 Mean + bi             0.944  0.891  0.738
## 5 Mean + bi + bu        0.866  0.750  0.669
## 6 Regularized bi and bu 0.865  0.749  0.670
```

The Regularization of the linear model gave approximately the same RMSE value than the project RMSE.

Model 4

##Matrix Factorization with recosystem

Recosystem is a package for recommender system using matrix factorization[2]. It is typically used to approximate an incomplete matrix using the product of two matrices in a latent space. Other common names for this task include “collaborative filtering”, “matrix completion”, “matrix recovery”, etc. High performance multi-core parallel computing is supported in this package[3].

The Matrix Factorization process is conducted as below:

```
if(!require(recosystem))
  install.packagesif(!require(recosystem))

## Loading required package: recosystem

## Warning: package 'recosystem' was built under R version 4.0.4

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

## Warning: package 'recosystem' is in use and will not be installed

set.seed(1234, sample.kind = "Rounding") # This is a randomized algorithm

## Warning in set.seed(1234, sample.kind = "Rounding"): non-uniform
'Rounding'
## sampler used

# Convert the train and test sets into recosystem input format
train_data <- with(train.set, data_memory(user_index = userId,
                                          item_index = movieId,
                                          rating = rating))
test_data <- with(test.set, data_memory(user_index = userId,
                                        item_index = movieId,
                                        rating = rating))

# Convert validation sets to recosystem input format

validation_reco <- with(validation, data_memory(user_index = userId,
                                                item_index = movieId,
                                                rating = rating))
```

[2] A Matrix-factorization Library for Recommender Systems - https://www.csie.ntu.edu.tw/~cjlin/libmf/

[3] https://cran.r-project.org/web/packages/recosystem/index.html

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

# identifying the best tuning parameters
opts <- r$tune(train_data, opts = list(dim = c(10, 20, 30),
                                       lrate = c(0.1, 0.2),
                                       costp_l2 = c(0.01, 0.1),
                                       costq_l2 = c(0.01, 0.1),
                                       nthread = 4, niter = 10))
# Train the algorithm
r$train(train_data, opts = c(opts$min, nthread = 4, niter = 20))

## iter      tr_rmse          obj
##    0       0.9922   1.0008e+07
##    1       0.8785   8.0758e+06
##    2       0.8473   7.5029e+06
##    3       0.8264   7.1635e+06
##    4       0.8092   6.9157e+06
##    5       0.7955   6.7290e+06
##    6       0.7842   6.5867e+06
##    7       0.7746   6.4729e+06
##    8       0.7663   6.3761e+06
##    9       0.7594   6.3009e+06
##   10       0.7531   6.2330e+06
##   11       0.7476   6.1770e+06
##   12       0.7426   6.1271e+06
##   13       0.7382   6.0841e+06
##   14       0.7340   6.0453e+06
##   15       0.7303   6.0113e+06
##   16       0.7268   5.9792e+06
##   17       0.7237   5.9533e+06
##   18       0.7209   5.9286e+06
##   19       0.7183   5.9076e+06

# using the model for prediction
y_hat_reco <- r$predict(test_data, out_memory())
head(y_hat_reco, 10)

##  [1] 4.115337 5.057001 5.420639 4.859281 4.458977 4.494849 4.590529
4.603108
##  [9] 4.003280 2.629064

# validating the model predictions
y_hat_final_reco <- r$predict(test_data, out_memory())

#compute the results table from start to finish
result <- bind_rows(result,
                    tibble(Method = "Matrix Factorization using recosystem",
                           RMSE = RMSE(test.set$rating, y_hat_reco),
                           MSE = MSE(test.set$rating, y_hat_reco),
                           MAE = MAE(test.set$rating, y_hat_reco)))
result
```

```
## # A tibble: 7 x 4
##   Method                               RMSE    MSE    MAE
##   <chr>                               <dbl>  <dbl>  <dbl>
## 1 Project Goal                        0.865 NA     NA    
## 2 Random prediction                   1.50   2.25   1.17 
## 3 Mean                                1.06   1.12   0.855
## 4 Mean + bi                           0.944  0.891  0.738
## 5 Mean + bi + bu                      0.866  0.750  0.669
## 6 Regularized bi and bu               0.865  0.749  0.670
## 7 Matrix Factorization using recosystem 0.791  0.625  0.609
```

Matrix factorization among the models used above resulted in a better RMSE estimate (0.791) than both the project RMSE goal and the Regularized linear model. As it has been stated earlier, the more the RMSE value tends to 0, the better the precision of the model.

Validation of Selected model

So far, we have seen that regularization and matrix factorization gave us a satisfactory RMSE. Using these models, the complete edx will be trained and we use the validation set to derive the RMSE in the `validation` set. What we hope to achieve is that the RMSE value remains below the project RMSE goal for consistency.

Applying the validation set to the Regularization and the matrix factorization model:

Validation of the linear model Regularization

```
#running the model training again on the edx set

mu_edx <- mean(edx$rating)

# Edx Movie bias (bi)
b_i_edx <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_edx)/(n()+lambda))

# Edx User bias (bu)
b_u_edx <- edx %>%
  left_join(b_i_edx, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu_edx)/(n()+lambda))

# Prediction using validation
y_hat_edx <- validation %>%
  left_join(b_i_edx, by = "movieId") %>%
  left_join(b_u_edx, by = "userId") %>%
  mutate(pred = mu_edx + b_i + b_u) %>%
  pull(pred)

# Update the results table
result <- bind_rows(result,
                    tibble(Method = "Validating Regularized Linear Model",
                           RMSE = RMSE(validation$rating, y_hat_edx),
                           MSE  = MSE(validation$rating, y_hat_edx),
                           MAE  = MAE(validation$rating, y_hat_edx)))

# RMSE of the validation is:
result

## # A tibble: 8 x 4
##   Method                                RMSE    MSE    MAE
##   <chr>                                <dbl>  <dbl>  <dbl>
## 1 Project Goal                         0.865 NA     NA
## 2 Random prediction                    1.50   2.25   1.17
## 3 Mean                                 1.06   1.12   0.855
## 4 Mean + bi                            0.944  0.891  0.738
## 5 Mean + bi + bu                       0.866  0.750  0.669
## 6 Regularized bi and bu                0.865  0.749  0.670
## 7 Matrix Factorization using recosystem 0.791  0.625  0.609
## 8 Validating Regularized Linear Model   0.865  0.748  0.669
```

The RMSE calculated using the `validation` set on the edx set (0.8648201) is found to be approximately the target of 0.8649, this implies that the model is consistent.

Validating the Matrix Factorization Model

The entire edx is transformed to a reco dataset and `validation` set is tested on it to see if the RMSE is consistently below the project RMSE goal.

```
set.seed(1234, sample.kind = "Rounding")

# Transforming the 'edx' and 'validation' sets to recosystem datasets
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))

# Creating the reco model object
r <-  recosystem::Reco()

# Parameter Tuning
opts <-  r$tune(edx.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))

# Model Training
r$train(edx.reco, opts = c(opts$min, nthread = 4, niter = 20))

## iter      tr_rmse          obj
##    0       0.9719   1.2028e+07
##    1       0.8719   9.8814e+06
##    2       0.8392   9.1825e+06
##    3       0.8174   8.7605e+06
##    4       0.8013   8.4761e+06
##    5       0.7892   8.2765e+06
##    6       0.7794   8.1219e+06
##    7       0.7714   8.0028e+06
##    8       0.7643   7.9046e+06
##    9       0.7585   7.8232e+06
##   10       0.7534   7.7555e+06
##   11       0.7489   7.7014e+06
##   12       0.7448   7.6505e+06
##   13       0.7411   7.6073e+06
##   14       0.7377   7.5691e+06
##   15       0.7346   7.5330e+06
##   16       0.7317   7.5019e+06
##   17       0.7290   7.4726e+06
##   18       0.7266   7.4492e+06
##   19       0.7244   7.4277e+06

# Making the Prediction
y_hat_reco_valid <-  r$predict(validation.reco, out_memory())

# Update the result table
result <- bind_rows(result,
                    tibble(Method = "Final Matrix Factorization - 
Validation",
                           RMSE = RMSE(validation$rating, y_hat_reco_valid),
                           MSE  = MSE(validation$rating, y_hat_reco_valid),
                           MAE  = MAE(validation$rating, y_hat_reco_valid)))
result

## # A tibble: 9 x 4
##   Method                                 RMSE    MSE    MAE
##   <chr>                                 <dbl>  <dbl>  <dbl>
## 1 Project Goal                          0.865 NA     NA    
## 2 Random prediction                     1.50   2.25   1.17 
## 3 Mean                                  1.06   1.12   0.855
## 4 Mean + bi                             0.944  0.891  0.738
## 5 Mean + bi + bu                        0.866  0.750  0.669
## 6 Regularized bi and bu                 0.865  0.749  0.670
## 7 Matrix Factorization using recosystem   0.791  0.625  0.609
## 8 Validating Regularized Linear Model     0.865  0.748  0.669
## 9 Final Matrix Factorization - Validation 0.782  0.612  0.603
```

The validated RMSE from matrix factorization is 0.783 which is better than the initial models’ RMSE and project RMSE goal because it tends towards zero.

The Matrx Factorization method is hence the most appropriate model for the Movielens recommendation system.

CHAPTER 4

CONCLUSION AND RECOMMENDATION

Summary

As stated earlier, the main objective of this study is to estimate two or more models to select the model that results in the best RMSE values. The project started by giving backgroung to recommendation systems as used in other real life examples.

The study progressed to importing, cleaning and exploring the movielens data. Univariate and bivariate explorations were done using both estimates and vizualization methods.

The model estimation was carried out starting from a random prediction method to mean predictions and adjusting for both movie and user effects/bias. The regularization of the linear model was also carried out, and the study was concluded on the matrix factorization model using the recosystem package.

The project RMSE goal was given as 0.8649000. The Regularized Linear model gave an approximate RMSE to the project RMSE while the Matrix Factorization produced a much lower RMSE than the project and Regularized model RMSE. The Matrix Factorization model is hence most appropriate than other models tested in the study.

Limitations

The first limitation observed in this work is that it uses only the movie and user effects as predictors. Other features such as genre and date were discaded. It was observed while fitting the linear model, the more features are added, the better the RMSE values. Perharps the addition of more features may help the model.

The models reported are the models that eventually and successfully run on the regular pc or 4G RAM and 500G HDD, other models could not find enogh space to run while some others threatened to crash the pc. A more sophisticated pc is needed to run most ML models.

However, the structure of the model is limited to only existing users and movies. Whenever a new movie or user is introduced, the bias is recalculated and the whole model will be refitted. The limitation of the model is that it cannot take new movies and users.

Future Work

For those who might want to take on this project, it is highly recommended that an alienware pc is used or any high performance pc with very high precision and storage space. This might make it posible to include the genre effect and run successfully. Future work might also browse other recommender packages in R (not sure this is welcomed for this course).

References

1. Bickel Peter J and Li Bo (2006), recosystem: recommendation System Using Parallel Matrix Factorization

2. Jason Brownlee (2019), A Gentle Introduction to Matrix Factorization for Machine Learning

3. Jason Brownlee (2020), Machine Learning Mastery with R, Get started, Build Accurate Models, and Work through Projects step-by-step

4. Ong Cheng Soon (2005), Kernels: Regularization andOptimization

5. Rafael A. Irizarry (2019), Introduction to Data Science: Data Analysis and Prediction Algorithms with R

6. Vijay Kotu, Bala Deshpande, (2019), [Recomendation Ssystem: Matrix Factorization] (https://www.sciencedirect.com/topics/computer-science/matrix-factorization)

7. Yixuan Qiu, et al., (2020), recosystem: recommendation System Using Parallel Matrix Factorization