MovieLensProject

Idxian D Gonzalez

11/27/2021

Introduction

This project is developed as part as a requirement for the Data Science: Capstone course, this is the final course of the EdX Data Science professional certificate. The purpose of this report is to apply the concepts and methods learned through the eight courses journey. *MovieLens 10M dataset* was used to generate this report.

The *MovieLens Dataset* was developed in 1997 by researchers of the University of Minnesota, with the aim of describing people's preferences for movies of distinct genres. GroupLens is the organization in charge of the page where this data is provided. This dataset is widely used for academic purposes as well as in the film industry and has undergone several changes in its structure over the years (Harper & Konstan, 2015).

The specific aim of this project is to predict movie ratings and determine the root mean square estimates [RMSE] score to evaluate the success of the prediction. RMSE is a statistical metric to measure a model performance through a score that indicates how accurate is the prediction made. A lower RMSE score indicates more accuracy from the model. The goal of this project is to obtain an RMSE score < 0.86490.

```
##Making sure required packages are available##
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 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.0     v forcats 0.5.1
```

```
## -- 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

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

## Loading required package: lubridate

##
## Attaching package: 'lubridate'

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

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union

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

## Loading required package: Metrics

##
## Attaching package: 'Metrics'

## The following objects are masked from 'package:caret':
##
##     precision, recall
```

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

##Setting libraries##
library(tidyverse)
library(caret)
library(lubridate)
library(data.table)
library(Metrics)
library(tidyr)

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

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

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

colnames(movies) <- c("movieId","title","genres")

##Since I'm using R 4.0.5##
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)
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)
```

Methods/analysis

Data exploration

MovieLens 10M dataset was divided into two datasets: *edx* & *validation*. The *edx* dataset consists of 90% of the data, while the *validation* dataset consists of 10% of the remaining data. We will be making our predictions using the *edx* dataset.The composition of both sets is the following:

```
str(edx) ## Evaluate Edx Set
```

```
## 'data.frame':    9000061 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
```

```
str(validation) ## Evaluate Validation Set
```

```
## 'data.frame':    999993 obs. of  6 variables:
##  $ userId   : int  1 2 2 3 3 3 4 4 4 5 ...
##  $ movieId  : num  588 1210 1544 151 1288 ...
##  $ rating   : num  5 4 3 4.5 3 3 3 3 5 3 ...
##  $ timestamp: int  838983339 868245644 868245920 1133571026 1133571035 1164885617 844416656 84441707
##  $ title    : chr  "Aladdin (1992)" "Star Wars: Episode VI - Return of the Jedi (1983)" "Lost World:
##  $ genres   : chr  "Adventure|Animation|Children|Comedy|Musical" "Action|Adventure|Sci-Fi" "Action|Ac
```

```
head(edx) ## See top Rows
```

```
##   userId movieId rating timestamp                         title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 3      1     231      5 838983392          Dumb & Dumber (1994)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
##                          genres
## 1                Comedy|Romance
## 2         Action|Crime|Thriller
## 3                        Comedy
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
```

The timestamp variable seems to be in a strange format, we can change it to a datetime format with the following code:

```
edx <-  edx %>% mutate (timestamp = as_datetime(timestamp)) ## changes date into a readable format

head(edx)
```

```
##   userId movieId rating           timestamp                         title
## 1      1     122      5 1996-08-02 11:24:06              Boomerang (1992)
## 2      1     185      5 1996-08-02 10:58:45               Net, The (1995)
## 3      1     231      5 1996-08-02 10:56:32          Dumb & Dumber (1994)
## 4      1     292      5 1996-08-02 10:57:01               Outbreak (1995)
## 5      1     316      5 1996-08-02 10:56:32               Stargate (1994)
## 6      1     329      5 1996-08-02 10:56:32 Star Trek: Generations (1994)
##                          genres
## 1                Comedy|Romance
## 2          Action|Crime|Thriller
## 3                        Comedy
## 4  Action|Drama|Sci-Fi|Thriller
## 5       Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
```

Now we can look at the *edx* data set and see that the date is in a much friendlier format. The movie year appears to be embedded inside the movie title. We can extract the year with the following code:

```
edx <-  edx %>% mutate (year = substring(title, nchar(title) - 6 )) ## Extracting Year from title name

head(edx)
```

```
##   userId movieId rating           timestamp                         title
## 1      1     122      5 1996-08-02 11:24:06              Boomerang (1992)
## 2      1     185      5 1996-08-02 10:58:45               Net, The (1995)
## 3      1     231      5 1996-08-02 10:56:32          Dumb & Dumber (1994)
## 4      1     292      5 1996-08-02 10:57:01               Outbreak (1995)
## 5      1     316      5 1996-08-02 10:56:32               Stargate (1994)
## 6      1     329      5 1996-08-02 10:56:32 Star Trek: Generations (1994)
##                          genres    year
## 1                Comedy|Romance  (1992)
## 2          Action|Crime|Thriller  (1995)
## 3                        Comedy  (1994)
## 4  Action|Drama|Sci-Fi|Thriller  (1995)
## 5       Action|Adventure|Sci-Fi  (1994)
## 6 Action|Adventure|Drama|Sci-Fi  (1994)
```

This more or less worked, however there are still parentheses in the data. We can remove those with the following code:

```
edx$year <- gsub("[()]", "", edx$year) ## Removing parenthesis
```

This worked, but seeing as year was a string we just manipulated, its probably not in a numeric format where we can properly process it. Lets fix this:

```
edx$year <- as.numeric(edx$year ) ## Coerce string into number
str(edx)
```

```
## 'data.frame':    9000061 obs. of  7 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 364 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
```

```
##  $ timestamp: POSIXct, format: "1996-08-02 11:24:06" "1996-08-02 10:58:45" ...
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
##  $ year     : num  1992 1995 1994 1995 1994 ...
```

It will also be necessary to split the genres by movie, as a movie can have multiple genres separated by pipes:

```
edx <- edx %>% separate_rows(genres, sep = "\\|")
## use separate_rows function split genres into single values.
##Movies with multiple genres will be duplicated in the dataset
head(edx)
```

```
## # A tibble: 6 x 7
##   userId movieId rating timestamp           title                genres    year
##    <int>   <dbl>  <dbl> <dttm>              <chr>                <chr>    <dbl>
## 1      1     122      5 1996-08-02 11:24:06 Boomerang (1992)     Comedy    1992
## 2      1     122      5 1996-08-02 11:24:06 Boomerang (1992)     Romance   1992
## 3      1     185      5 1996-08-02 10:58:45 Net, The (1995)      Action    1995
## 4      1     185      5 1996-08-02 10:58:45 Net, The (1995)      Crime     1995
## 5      1     185      5 1996-08-02 10:58:45 Net, The (1995)      Thriller  1995
## 6      1     231      5 1996-08-02 10:56:32 Dumb & Dumber (1994) Comedy    1994
```

Finally, we will need to make the same transformations to the validation file to keep everything equal

```
validation <-  validation %>% mutate (timestamp = as_datetime(timestamp))
## Repeat all steps for validation set
validation <-  validation %>% mutate (year = substring(title, nchar(title) - 6 ))
## Extract Year
validation$year <- gsub("[()]", "", validation$year) ## Remove Parens
validation$year <- as.numeric(validation$year ) ## coerce Numeric
validation <- validation %>% separate_rows(genres, sep = "\\|")
## use separate_rows function split genres into single values. Movies with multiple
## genres will be duplicated in the dataset

str(validation)
```

```
## tibble [2,597,587 x 7] (S3: tbl_df/tbl/data.frame)
##  $ userId   : int [1:2597587] 1 1 1 1 1 2 2 2 2 2 ...
##  $ movieId  : num [1:2597587] 588 588 588 588 588 ...
##  $ rating   : num [1:2597587] 5 5 5 5 5 4 4 4 3 3 ...
##  $ timestamp: POSIXct[1:2597587], format: "1996-08-02 10:55:39" "1996-08-02 10:55:39" ...
##  $ title    : chr [1:2597587] "Aladdin (1992)" "Aladdin (1992)" "Aladdin (1992)" "Aladdin (1992)" ..
##  $ genres   : chr [1:2597587] "Adventure" "Animation" "Children" "Comedy" ...
##  $ year     : num [1:2597587] 1992 1992 1992 1992 1992 ...
```

Data visualization

All the visualizations and analysis will be made using the *edx* data set. *edx* variable classes are:

```
sapply(edx,class) ## Validate class of variables
```

```
## $userId
## [1] "integer"
##
## $movieId
## [1] "numeric"
##
## $rating
## [1] "numeric"
##
## $timestamp
## [1] "POSIXct" "POSIXt"
##
## $title
## [1] "character"
##
## $genres
## [1] "character"
##
## $year
## [1] "numeric"
```

Genres

Having successfully loaded and wrangled the data, we can create a summary dataframe to examine the distribution of genres across movies. We can do so with the following code:

```
df <- edx %>% group_by(genres) %>% summarise(n = n()) ## See top N genre distribution
```

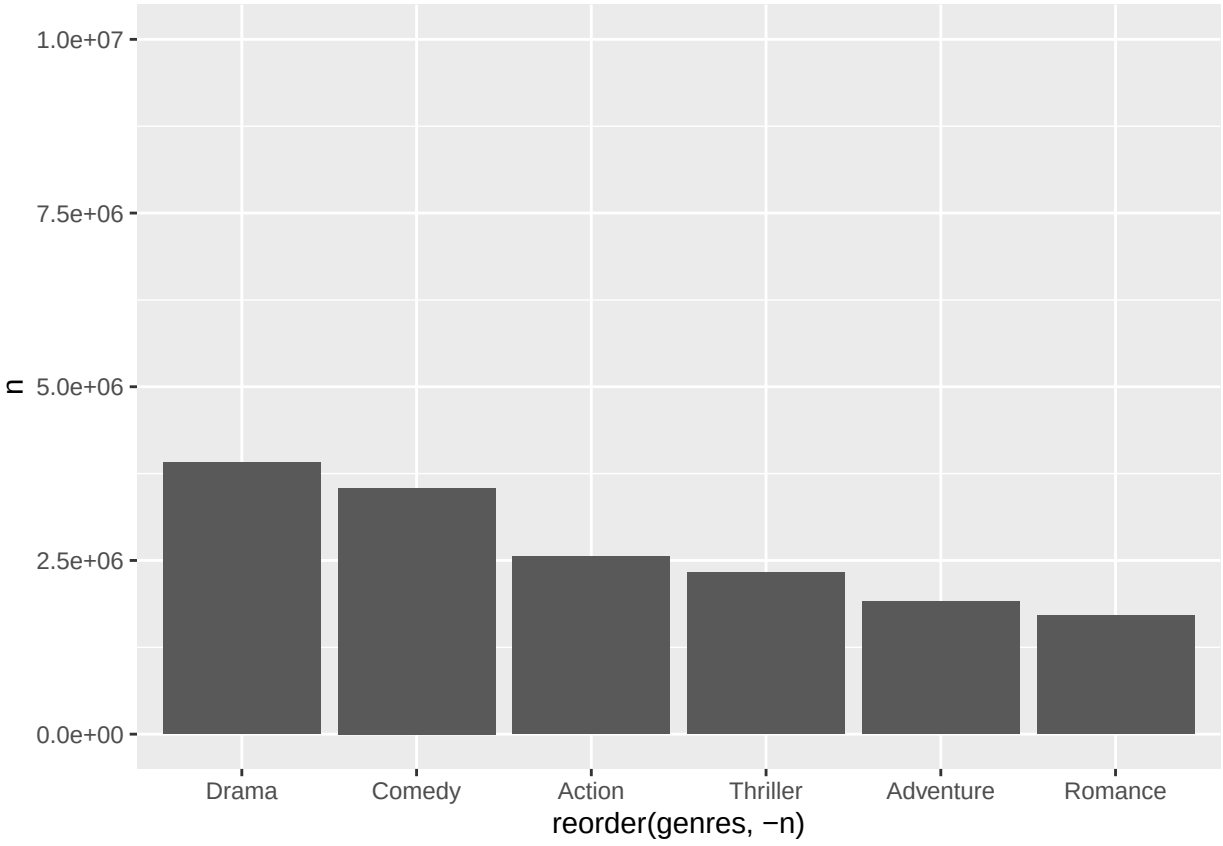
We can also examine the distribution of genres by frequency:

```
head(df[order(-df$n),]) ## Sort by tops
```

```
## # A tibble: 6 x 2
##   genres          n
##   <chr>       <int>
## 1 Drama     3909401
## 2 Comedy    3541284
## 3 Action    2560649
## 4 Thriller  2325349
## 5 Adventure 1908692
## 6 Romance   1712232
```

We can observe that the top 3 genres are Drama, Comedy and Action. We can further validate this by looking at the frequency distribution of the Genre variable:

```
head(df[order(-df$n),]) %>% ## Feed reduced table for graphing
ggplot(aes(x = reorder(genres, -n), n)) +
geom_bar(stat="identity") + scale_y_continuous(limits=c(0,10000000)) ## Create plot to see top genre
```

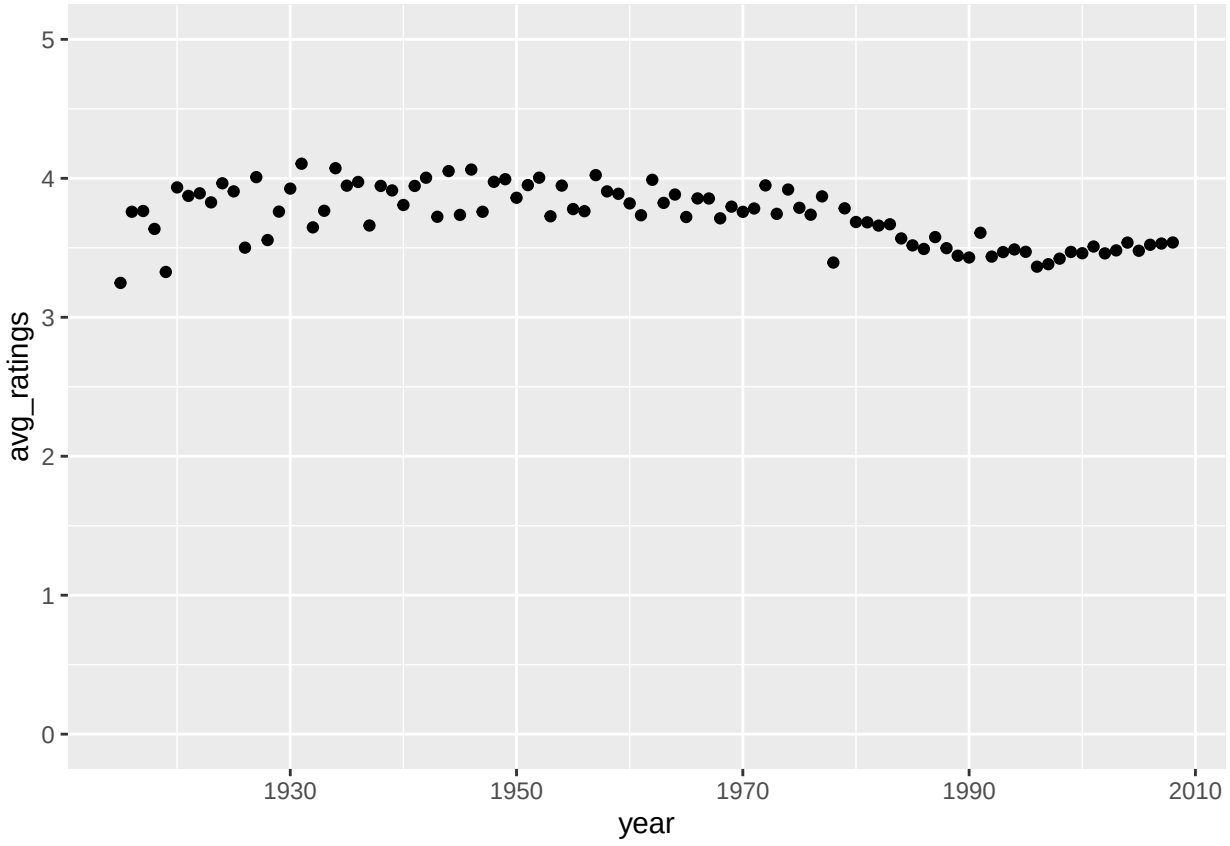


Ratings

In *MovieLens* the ratings are expressed as a star value. Literature suggests that this variable experienced a change by 2003, when the rating classification system shifted from a whole star to half stars ratings (Harper & Konstan,2015,p.8). A new dataframe was developed in order to obtain the average rating per year.

The following plot shows that until the 1980's the variability of the data was higher, but by 1990 the variability seems to be less. This suggests that the variable year might be an important feature in modeling for ratings, since variable dispersion seems to decrease the closer we get to the present. We can see this is in the following plot:

```
edxavg_rating <- edx %>%  group_by(year) %>%
  summarize(avg_ratings = mean(rating,trim = 0,na.rm = FALSE))
## create DataFrame with average ratings by year
edxavg_rating %>% ggplot(aes(x = year,y = avg_ratings)) +
geom_point() + scale_y_continuous(limits=c(0,5)) ## plot average ratings per year
```



Results

Having evaluated these variables, we can begin constructing our model to predict ratings. Our goal for this task is to generate a model that can minimize the RMSE for this particular data set. We will be using the *edx* data set to train the model, and validating it's input with the *validation* dataset. We will be using the RMSE function from the Metrics package for this:

```
avg <- mean(edx$rating)  # Mean movie rating in EDX set
RMSECalc <- rmse(avg,validation$rating) # RMSE calculations
RMSECalc #Display RMSE
```

```
## [1] 1.052443
```

This model gives us an RMSE of 1.052, which means that the average prediction would be almost 1.05 stars away from the actual value. Maybe we can get this down further by utilizing more variables in our model. We can evaluate the effect of year added to this average with the following code:

```
year <- edx %>%
  group_by(year) %>% # Group Data by year
  summarize(diff = mean(rating - avg)) # calculate difference between average value and actual value


ratingsbyyear <- validation %>%  left_join(year, by = "year") %>% # Join on Year
  mutate(ratingprediction =  diff + avg) %>% # Generate new predictions
  pull(ratingprediction)  # retrieve
```

```
rmse_Year <- rmse(ratingsbyyear,validation$rating)  # calculate RMSE for model

rmse_Year
```

```
## [1] 1.042132
```

This model gives us an RMSE of 1.042, which is lower than our average model but not by much. We can try the same process with the *UserId* variable to validate the effect over the general average:

```
user <- edx %>%
  group_by(userId) %>% # Group Data by UserId
  summarize(diff = mean(rating - avg)) # calculate difference between average value and actual value


ratingsbyUser <- validation %>%  left_join(user, by = "userId") %>% # Join on UserId
  mutate(ratingprediction =  diff + avg) %>% # Generate new predictions
  pull(ratingprediction)  # retrieve

rmse_User <- rmse(ratingsbyUser,validation$rating)  # calculate RMSE for model

rmse_User
```

```
## [1] 0.9732766
```

Taking into account the *UserId* as a variable, we can get a RMSE of 0.973, which is considerably lower than the previous 1.05. This variable thus far seems to have the most effect over the rating variable. Finally, we can try the *MovieId* variable as a predictor on top of the average with the following code:

```
movieId <- edx %>%
  group_by(movieId) %>% # Group Data by movieId
  summarize(diff = mean(rating - avg)) # calculate difference between average value and actual value


ratingsbymovie <- validation %>%  left_join(movieId, by = "movieId") %>% # Join on movieId
  mutate(ratingprediction =  diff + avg) %>% # Generate new predictions
  pull(ratingprediction)  # retrieve

rmse_movie <- rmse(ratingsbymovie,validation$rating)  # calculate RMSE for model

rmse_movie
```

```
## [1] 0.9411063
```

This model has an RMSE of 0.941, which is again lower than our best estimate of 0.97. After our initial data visualizations seemed to suggest that the *year* variable was important in predicting a movie rating, our RMSE estimates show that the most impactful variables on ratings are the *UserId* and the *MovieId*. We can explore both of them at the same time with the following code:

```
movie <- edx %>% group_by(movieId) %>% summarize(diff1 = mean (rating - avg)) ## Calculate movie effect

user <-  edx %>% left_join(movie, by = "movieId") %>% group_by(userId) %>%
```

```
summarize(diff2 = mean(rating - avg - diff1)) #calculate user effect, given movie effect

pred <- validation %>% left_join(movie, by = "movieId") %>%
  left_join(user, by = "userId") %>% mutate(prediction = avg + diff1 + diff2) %>% pull(prediction)
# compare predicted values to validation dataset

rmse_user_movie <- rmse(validation$rating, pred) ## calculate RMSE

rmse_user_movie # View RMSE
```

```
## [1] 0.8635899
```

With this combined model, we get an RMSE of 0.8635, which is notably lower than all our individual predictions. As such, we conclude this assignment by generating a model which provides a RMSE of 0.8635 by utilizing the variables *UserId* and *MovieId* as predictors.

Conclusion

MovieLens 10M dataset was analyzed, the variables *MovieId* and *UserId* seems to be meaningful in the data prediction model. When we develop a model considering both we get a lower RMSE score.

Specifically the final RMSE achieved in the model was:

```
rmse_user_movie
```

```
## [1] 0.8635899
```

Limitations

Regularization of the data was excluded from this analysis, further wrangling of the data could yield more precise models.

Future Work

Further analysis should consider other variables, this could lead to the improvement of the model.

References

Harper, F. M., & Konstan, J. A. (2015). The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis), 5*(4), 1-19.