MovieLens Capstone Project

Justin Oh

08/13/2022

Introduction

Netflix, currently the most popular streaming service in the world, launched a competition in 2006 promising $1 million in prize money for the winning team. The objective of the challenge was to create an algorithm that outperformed Netflix's baseline algorithm, Cinematch, by 10%. The "Netflix Prize" contest highlighted the significance and financial benefits of research focused on improving existing recommendation systems, with companies such as social media firm Twitter hosting similar competitions of their own.

In the HarvardX: PH125.9x Data Science: Capstone course, an objective comparable to the "Netflix Prize" contest is presented to participants, albeit with the use of a different dataset. Rather than the datasets provided by Netflix, the 10M version of the MovieLens dataset provided by the research lab GroupLens will be used. As the name suggests, the dataset contains 10 million movie ratings, as well as 100,000 tag applications applied to 10,000 movies by 72,000 users.

Goal of the Project

The MovieLens Project is designed to encompass the various machine learning and computational concepts learned from previous courses in the HarvardX Professional Data Science Certificate program. Using the MovieLens dataset provided, the goal of the project is to train an algorithm with a RMSE score less than 0.86490.

The Root Mean Square Error, or RMSE, is the value used to assess the algorithm's performance. Generally, a lower RMSE score indicates better performance, and vice versa.

The following function computes the RMSE based on predicted and observed values:

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

Key Steps Performed

The MovieLens Project consisted of four key stages, and as such, the report has been divided into four distinct sections for improved readability.

1) Introduction: The context needed to understand the MovieLens Project is provided such as the definition of RMSE. Proceeded to download the MovieLens 10M dataset and created the "edx" and "valiation" subsets, which will serve as the training dataset and final hold-out test dataset, respectively.
2) Methods/Analysis: Performed exploratory analysis using the MovieLens dataset to clarify the influence that each feature has on movie ratings.

3) Results: Created multiple models that incorporate varying features of the MovieLens dataset while presenting the code associated. Upon doing so, the RMSE value of each model was calculated in search of the best performing algorithm.
4) Conclusion: Provided a brief summary of my MovieLens Project as a whole, as well as conclusions that could be made hrough the results from the models created. Limitations specific to my project are elaborated on and future work is discussed for closure.

Overview of Provided Code

Below is the code provided by HarvardX:

```
############################################################
# Create edx set, validation set (final hold-out test set)
############################################################
# Note: this process could take a couple of minutes
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
library(tidyverse)
library(caret)
library(data.table)
# 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 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")
```

As mentioned previously, the MovieLens dataset will be split into 2 subsets: a training subset (edX) and a testing subset (Validation).

```
# 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)`
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)
edx <- rbind(edx, removed)
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Examining the Data

After the data is loaded, we will examine the structure, classifications and corresponding statistics of our data set for better understanding of the source data.

We will first check to see whether there are any missing values both the "edx" and "validation" subsets.

```
#check any missing value in training data set "edx"
anyNA(edx)
```

```
## [1] FALSE
```

```
#check any missing value in final hold-out test data set "validation"
anyNA(validation)
```

```
## [1] FALSE
```

```
#list amount of observations and variables in training data set "edx"
dim(edx)
```

```
## [1] 9000055       6
```

```
#list amount of observations and variables in final hold-out test data set "validation"
dim(validation)
```

```
## [1] 999999      6
```

The results above show that neither the data set "edx" nor the "validation" include any incorrect or missing values. We can see that the "edx" data set has a total of 9,000,055 observations with 6 features, whereas the "validation" data set contains 999,999 observations with 6 features. We now need to be aware of their various statistics and distinguishable qualities.

To learn more about the data set "edx," we are now exploring it further.

Here are the first six observations from the data set "edx" so that we can get a quick overview.

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                         genres
## 1:                Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

Here is statistics of data set "edx".

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

As of right now, we are aware that the data set "edx" initially included six features: "userId," "movieId," "rating," "timestamp," "title," and "genres." In the meanwhile, we note that the feature's "title" contains year information; this information may need to be divided for additional analysis. Additionally, in order to further examine the impact of time on rating, "timestamp" may need to be converted to rated year. Moving on, we see that there are several "genres" of feature types for a single observation. To handle the impact of genres on ratings in the next data processing portion, we might need to divide "genres." We now know that the minimum rating is 0.5 and the average rating is 3.512.

Data Preprocessing

We will perform some minimal data pre-processing on both the "edx" and "validation" data sets before we go into further in-depth data analysis. As part of the data pre-processing steps, the feature "timestamp" is converted to the year rated, the year released is extracted from the feature "title", the age between the year rated and year released is calculated, and all 3 new features are added to the original data sets.

```
# convert timestamp to year rated and add it to edx
edx <- edx %>% mutate(year_rated = year(as_datetime(timestamp)))
# double check any invalid value after convertion of timestamp
unique(edx$year_rated)
```

```
##  [1] 1996 1997 2006 2005 2001 2003 1999 2000 2002 1998 2007 2008 2004 2009 1995
```

```
# extract the year released from title and add it to edx
edx <- edx %>% mutate(year_released = as.numeric(str_sub(title,-5,-2)))
# double check any invalid value of year released
unique(edx$year_released)
```

```
##  [1] 1992 1995 1994 1993 1991 1937 1970 1977 1990 1996 1971 1983 1989 1974 1967
## [16] 1984 1997 1985 2000 1982 2002 2003 2004 2005 1976 1972 1958 1942 1939 1941
## [31] 1950 1951 1979 1955 1962 1960 1980 1988 1975 1987 1981 1969 1998 1999 1986
## [46] 2001 1952 1959 1946 1940 1954 1949 1973 1948 1933 1931 1963 1922 1943 1944
## [61] 1957 1953 1965 1978 1964 1968 1966 1961 1956 1935 1938 2006 2007 1932 2008
## [76] 1934 1945 1947 1927 1925 1930 1936 1929 1923 1928 1921 1926 1915 1924 1916
## [91] 1917 1918 1920 1919
```

```
# calculate the movie age when movie was rated and add it to edx
edx <- edx %>% mutate(ages = as.numeric(year_rated)-as.numeric(year_released))
# double check any invalid value of ages
unique(edx$ages)
```

```
##  [1]  4  1  2  3  5 59 26 20  7 14 11 10 12 16 31 38 21 13  8  9  6 17 24 15 25
## [26] 39 55 58 56 47 46 18 42 35 37 23 30 29 32  0 51 45 44 62 64 57 63 49 41 54
## [51] 43 19 70 72 36 34 40 81 28 52 60 22 50 53 27 33 66 67 61 69 68 48 65 74 75
## [76] 71 77 73 79 76 83 78 85 80 84 91 82 90 89 88 86 87 -1 93 92 -2
```

We can see that the pre-processing of the data produced decent results with the exception of the observation of 2 odd ages (-1 and -2). We're not sure if the movies were rated before they were officially released or if there were data issues. We need to further investigate the observations that have these peculiar values.

```
sum(edx$ages == -1)/nrow(edx)
```

```
## [1] 1.911099e-05
```

```
sum(edx$ages == -2)/nrow(edx)
```

```
## [1] 3.333313e-07
```

The number of observations with odd values are not significant enough to influence modeling to a concerning degree, so we will keep these values for now.

We will apply the same process on the "validation" dataset as well.

```
# do the same data pre-processing for validation set
validation <- validation %>% mutate(year_rated = year(as_datetime(timestamp)))
validation <- validation %>% mutate(year_released = as.numeric(str_sub(title,-5,-2)))
validation <- validation %>% mutate(ages = as.numeric(year_rated)-as.numeric(year_released))
```

Methods and Analysis

Data Analysis

Six variables — "userID," "movieID," "rating," "timestamp," "title," and "genres" — are included in the edx subset. Each row corresponds to a single user's rating of a particular movie.

```
##   userId movieId rating timestamp                          title
## 1      1     122      5 838985046                Boomerang (1992)
## 2      1     185      5 838983525                 Net, The (1995)
## 3      1     292      5 838983421                 Outbreak (1995)
## 4      1     316      5 838983392                 Stargate (1994)
## 5      1     329      5 838983392 Star Trek: Generations (1994)
## 6      1     355      5 838984474       Flintstones, The (1994)
##                         genres year_rated year_released ages
## 1                Comedy|Romance       1996          1992    4
## 2         Action|Crime|Thriller       1996          1995    1
```

```
## 3  Action|Drama|Sci-Fi|Thriller       1996          1995    1
## 4       Action|Adventure|Sci-Fi       1996          1994    2
## 5 Action|Adventure|Drama|Sci-Fi       1996          1994    2
## 6       Children|Comedy|Fantasy       1996          1994    2
```

Using the summarize function, we can check the edx subset for any missing values.

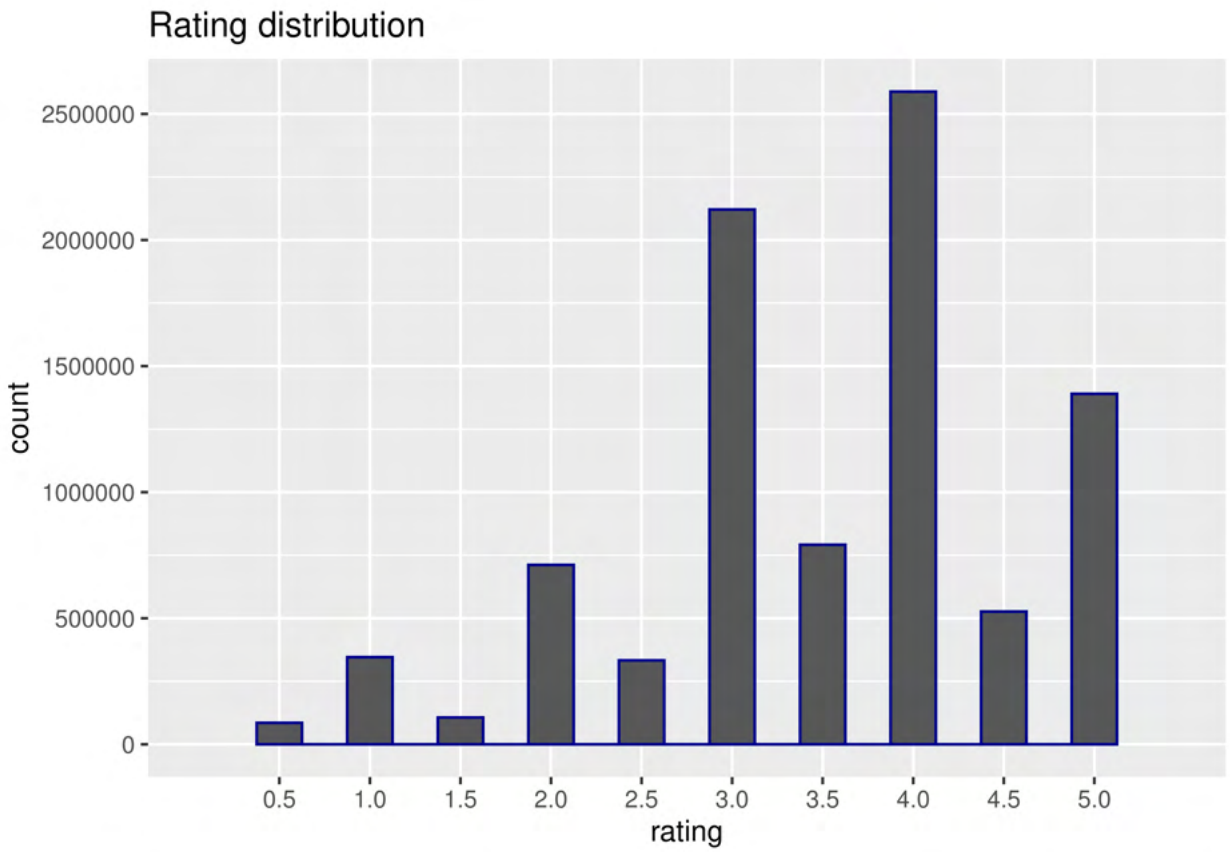
```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres           year_rated   year_released
##  Length:9000055     Length:9000055     Min.   :1995   Min.   :1915
##  Class :character   Class :character   1st Qu.:2000   1st Qu.:1987
##  Mode  :character   Mode  :character   Median :2002   Median :1994
##                                        Mean   :2002   Mean   :1990
##                                        3rd Qu.:2005   3rd Qu.:1998
##                                        Max.   :2009   Max.   :2008
##       ages
##  Min.   :-2.00
##  1st Qu.: 2.00
##  Median : 7.00
##  Mean   :11.98
##  3rd Qu.:16.00
##  Max.   :93.00
```

We can also calculate the number of unique movies and users in the edx subset:

```
##   n_users n_movies
## 1   69878    10677
```

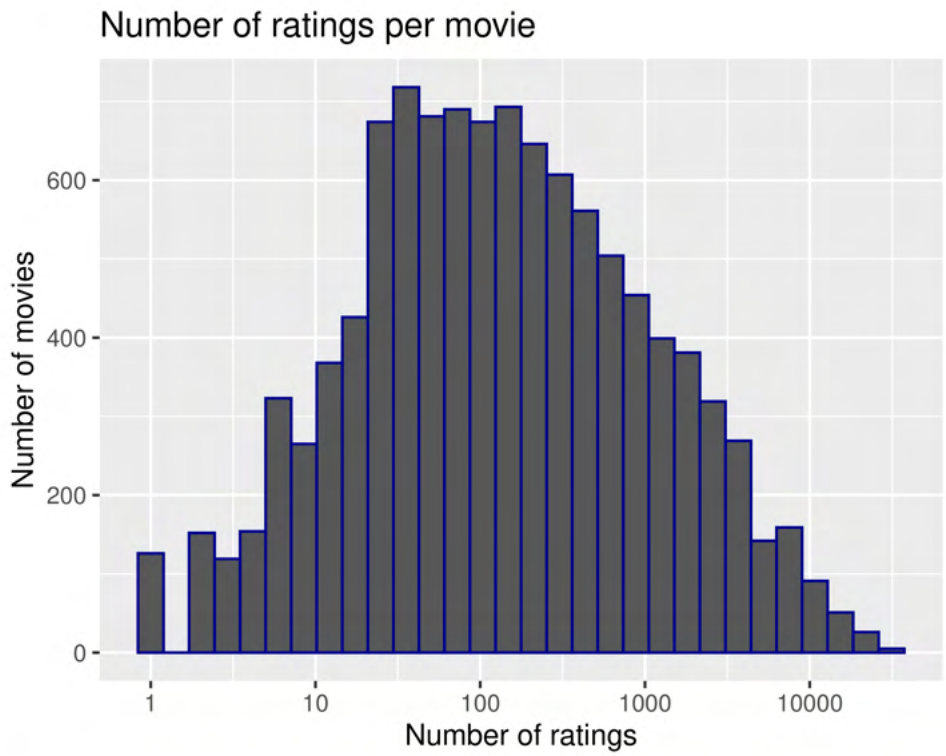
The figure below visually illustrates the left-skewed distribution of the rating scale. Users prefer to give movies better ratings than lower ones. The most prevalent rating is 4, followed by 3 and 5.

```
## Warning: Continuous limits supplied to discrete scale.
## Did you mean `limits = factor(...)` or `scale_*_continuous()`?
```



We can visualize the data and see that some films receive ratings more frequently than others, while other films receive extremely few or even no ratings at all. The models in this report will be subjected to regularization and a penalty term as a result.

```
edx %>%
count(movieId) %>%
ggplot(aes(n)) +
geom_histogram(bins = 30, color = "blue4") +
scale_x_log10() +
xlab("Number of ratings") +
  ylab("Number of movies") +
ggtitle("Number of ratings per movie")
```



It will be challenging to anticipate future ratings for the 20 films that have only received one rating given their obscurity.

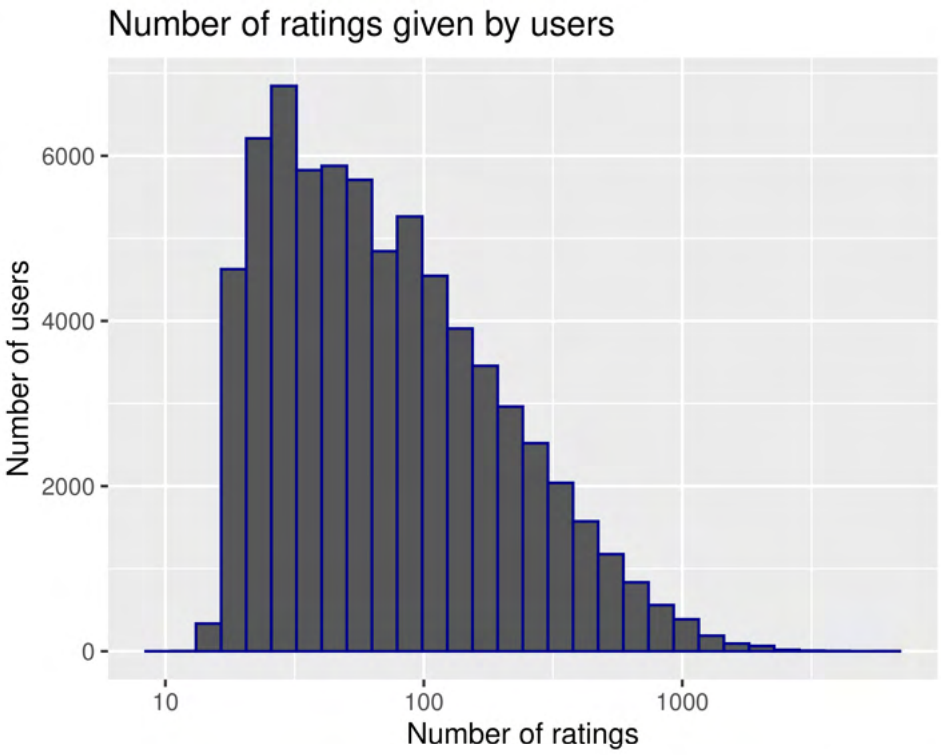
```
edx %>%
  group_by(movieId) %>%
  summarize(count = n()) %>%
  filter(count == 1) %>%
  left_join(edx, by = "movieId") %>%
  group_by(title) %>%
  summarize(rating = rating, n_rating = count) %>%
  slice(1:20) %>%
  knitr::kable()
```

title	rating	n_rating
1, 2, 3, Sun (Un, deuz, trois, soleil) (1993)	2.0	1
100 Feet (2008)	2.0	1
4 (2005)	2.5	1
Accused (Anklaget) (2005)	0.5	1
Ace of Hearts (2008)	2.0	1
Ace of Hearts, The (1921)	3.5	1
Adios, Sabata (Indio Black, sai che ti dico: Sei un gran figlio di...) (1971)	1.5	1
Africa addio (1966)	3.0	1
Aleksandra (2007)	3.0	1
Bad Blood (Mauvais sang) (1986)	4.5	1
Battle of Russia, The (Why We Fight, 5) (1943)	3.5	1
Bellissima (1951)	4.0	1
Big Fella (1937)	3.0	1
Black Tights (1-2-3-4 ou Les Collants noirs) (1960)	3.0	1

title	rating	n_rating
Blind Shaft (Mang jing) (2003)	2.5	1
Blue Light, The (Das Blaue Licht) (1932)	5.0	1
Borderline (1950)	3.0	1
Brothers of the Head (2005)	2.5	1
Chapayev (1934)	1.5	1
Cold Sweat (De la part des copains) (1970)	2.5	1

The vast majority of people have given between 30 to 100 movie ratings, and as a result, a user penalty term will be included in the models.

```
edx %>%
count(userId) %>%
ggplot(aes(n)) +
geom_histogram(bins = 30, color = "blue4") +
scale_x_log10() +
xlab("Number of ratings") +
ylab("Number of users") +
ggtitle("Number of ratings given by users")
```

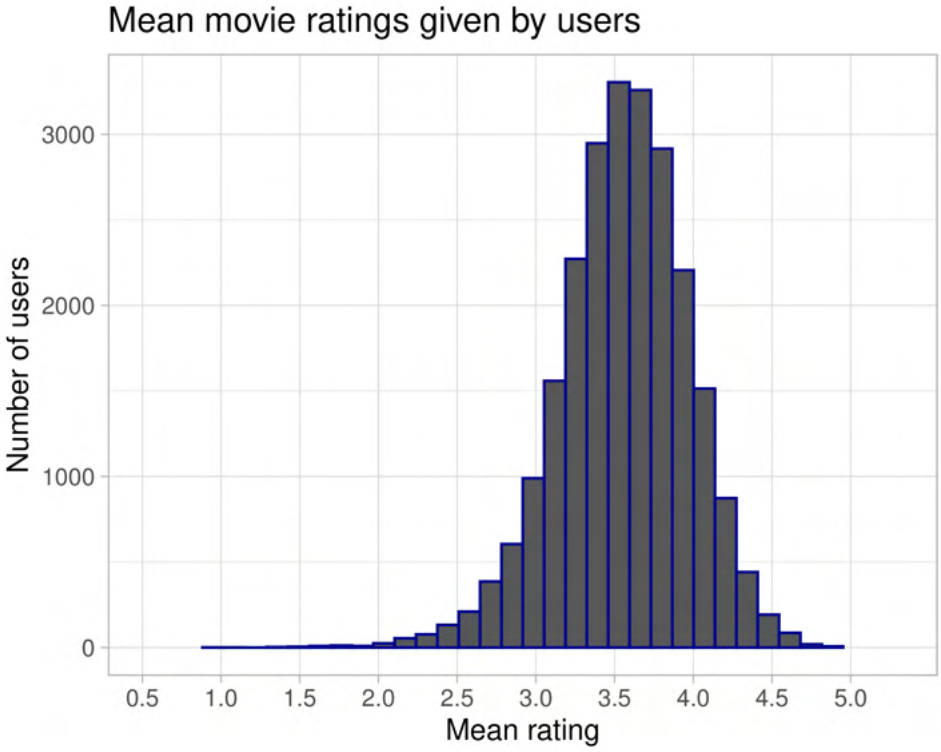


As we can see, the degree to which people are critical of a movie (which is reflected in their rating) also varies greatly. Some people have a tendency to rate items with far less stars than the typical user, and vice versa. Only users with at least 100 movie ratings are shown in the following graphic below.

```
edx %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
```

```
  summarize(b_u = mean(rating)) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "blue4") +
  xlab("Mean rating") +
  ylab("Number of users") +
  ggtitle("Mean movie ratings given by users") +
  scale_x_discrete(limits = c(seq(0.5,5,0.5))) +
  theme_light()
```

```
## Warning: Continuous limits supplied to discrete scale.
## Did you mean `limits = factor(...)` or `scale_*_continuous()`?
```



Modelling Approach

We write function, previously anticipated, that compute the RMSE, defined as follows:

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

N represents the number of user/movie combinations and the sum occurring over all these combinations. Our metric for model precision is the RMSE.

The following written function computes the RMSE through the vectors of ratings and their corresponding predictions:

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

The RMSE will judge the model's level of quality. As mentioned before, a lower RMSE score indicates better results.

1) Naive Model

We will develop a prediction system that just makes use of the sample mean. This suggests that each prediction is the sample average.

This model-based approacb assumes that all movies will receive the same rating and that any discrepancies will be due to random variation:

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

with $\epsilon_{u,i}$ independent error sample from the same distribution centered at 0 and μ the "true" rating for all movies. This very simple model makes the assumption that all differences in movie ratings are explained by random variation alone. We know that the estimate that minimizes the RMSE is the least square estimate of $Y_{u,i}$, which is the average of all ratings:

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

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

Here, we solve for the first naive RMSE. The resulting RMSE using this approach is higher than we would like.

```
naive_rmse <- RMSE(validation$rating, mu)
naive_rmse
```

```
## [1] 1.061202
```

Here, we represent results table with the first RMSE:

```
rmse_results <- data_frame(method = "Naive model", RMSE = naive_rmse)
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

```
rmse_results %>% knitr::kable()
```

method	RMSE
Naive model	1.061202

While this is far higher than the 0.86490 RMSE objective, this provides us with our baseline RMSE for comparison with future modeling strategies.

2) Movie Effect Model

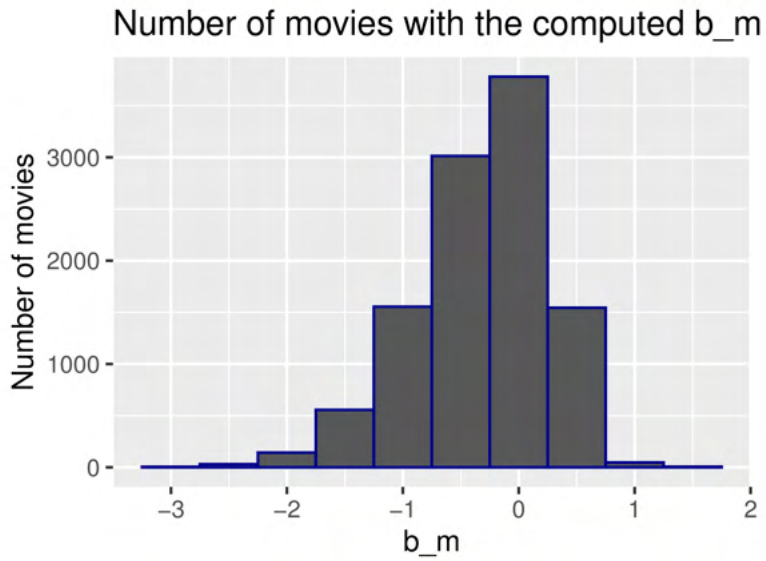
Popular movies typically receive higher ratings from people, whereas unpopular movies typically receive lower ratings.

We will compute the estimated deviation of each movies' mean rating from the total mean of all movies μ. The resulting variable is called "b" (as bias) for each movie "i" b_i, that represents average ranking for movie i:

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

The created histogram is left skewed, suggesting that more movies have negative effects.

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_m = mean(rating - mu))
movie_avgs %>% qplot(b_m, geom ="histogram", bins = 10, data = ., color = I("blue4"),
ylab = "Number of movies", main = "Number of movies with the computed b_m")
```



Our prediction improve once we predict using this model:

```
predicted_ratings <- mu +  validation %>%
  left_join(movie_avgs, by='movieId') %>%
  pull(b_m)
model_1_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie effect model",
                                     RMSE = model_1_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Naive model	1.0612018
Movie effect model	0.9439087

By adding the computed b_i to μ, we are able to predict movie ratings. If an individual movie is on average rated worse that the average rating of all movies μ , we predict that it will rated lower that μ by b_i, the difference of the individual movie average from the total average.

3) Movie + User Effects Model

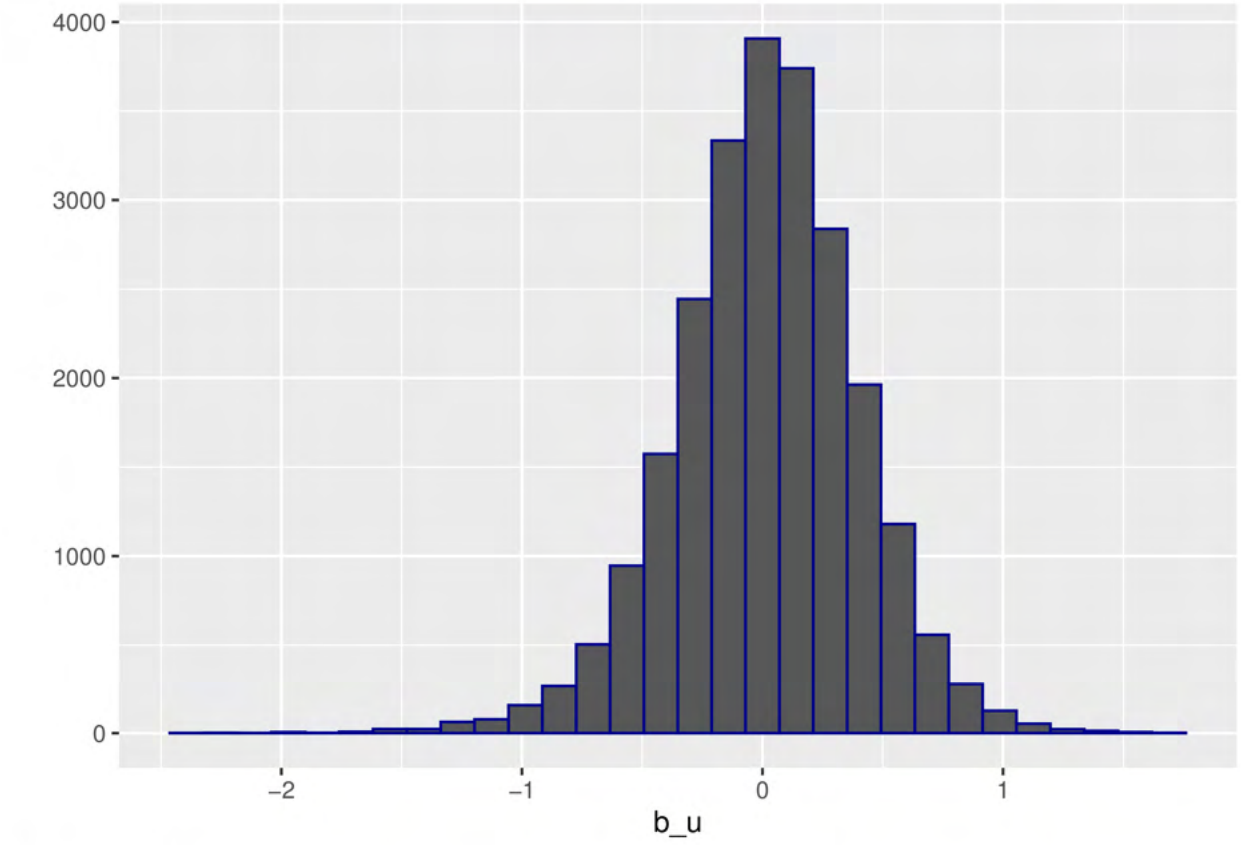
Users may have a tendency to rate movies higher or lower than the overall mean, so we can take this into account by incorporating it into the next model. First we will calculate the bias for each user:

$$b_u = Mean_{user} - \mu$$

Then we'll combine the bias of a user, with the bias of a film and add both to the overall mean for a combined bias rating for each unique combination of a user rating for a given film.

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

```
user_effects<- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  filter(n() >= 100) %>%
  summarize(b_u = mean(rating - mu - b_m))
user_effects%>% qplot(b_u, geom ="histogram", bins = 30, data = ., color = I("blue4"))
```



By computing μ and b_i, we can compute an approximation, and estimate b_u as the average of

$$Y_{u,i} - \mu - b_i$$

```
user_effects <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_m))
```

Now, we can create predictors and see how the RMSE improves:

```
predicted_ratings <- validation%>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_effects, by='userId') %>%
  mutate(prediction = mu + b_m + b_u) %>%
  pull(prediction)
model_2_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie and user effect model",
                                     RMSE = model_2_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Naive model	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488

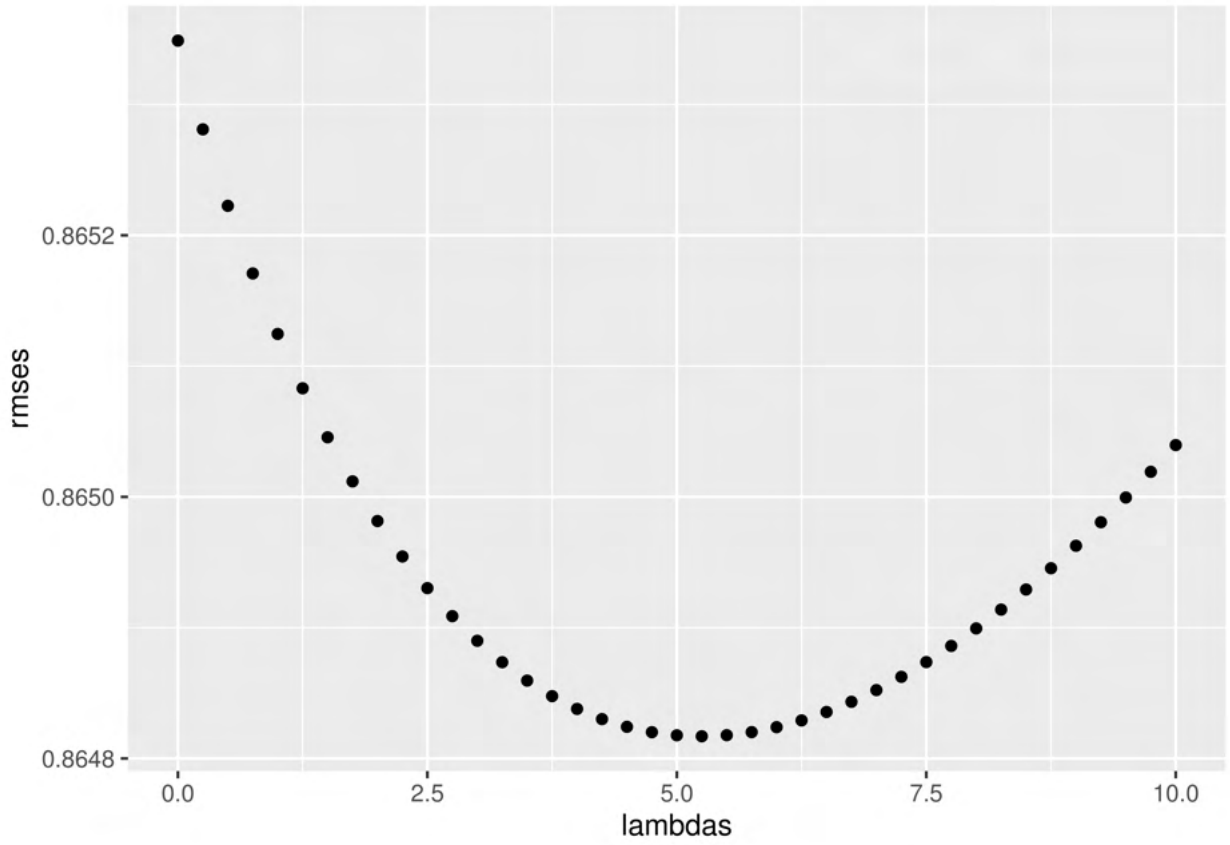
The Movie Effect Model was flawed as we only accounted for the "movieID" predictor, so with the addition of the "userID" predictor, our third model's rating predictions achieved a far lower RMSE value. Howeever, fewer users results in increased uncertainty of our predictions. Due to this, larger estimates of b_i, negative or positive, are more likely. To prevent these larger estimates from negatively impacting our RMSE, we will create a regularized model to avoid such circumstances.

4) Regularized Movie + User Effects Model

Regularization is typically used to limit the overall fluctuation, or variability, of the effect size. Lambda is a tuning parameter, so we can use cross-validation to select the best one for model construction and performance assessment.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
mu_reg <- mean(edx$rating)
#regulation movie effect
b_m_reg <- edx %>%
group_by(movieId) %>%
summarize(b_m_reg = sum(rating - mu_reg)/(n()+l))
#regulation user effect
b_u_reg <- edx %>%
left_join(b_m_reg, by="movieId") %>%
group_by(userId) %>%
summarize(b_u_reg = sum(rating - b_m_reg - mu_reg)/(n()+l))
#calculating predicted ratings
predicted_ratings_b_m_u <-
validation %>%
```

```
left_join(b_m_reg, by = "movieId") %>%
left_join(b_u_reg, by = "userId") %>%
mutate(prediction = mu_reg + b_m_reg + b_u_reg) %>%
.$prediction
return(RMSE(validation$rating,predicted_ratings_b_m_u))
})
qplot(lambdas, rmses)
```




```
#For the full model, the optimal lambda is given as
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 5.25
```

```
#calculating regularization model RMSE: model_m_u_reg_rmse
model_m_u_reg_rmse <- min(rmses)
model_m_u_reg_rmse
```

```
## [1] 0.864817
```

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Regularization Model",
                                     RMSE = model_m_u_reg_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Naive model	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488
Movie + User Regularization Model	0.8648170

Through the Regularized Movie + User Effects Model, we have finally achieved the target RMSE (less than 0.86490) with a RMSE value of 0.8648170.

Results

The table below shows the RMSE values achieved by all four models examined over the course of this report:

```
rmse_results %>% knitr::kable()
```

We drastically reduced the initial RMSE of 1.0612018 to 0.8648170 with the Regularized Movie + User Effect Model:

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

Conclusion

In order to construct a movie recommendation algorithm with a sufficiently low RMSE, I have employed an iterative applied machine learning approach. Beginning with a naive approach, I developed models that branched out to MovieLens features such as movie and user effects, and applied the regularization technique to further cut down the RMSE value.

Considering the fact that only two predictors, "movieId" and "userId", were used to obtain the desired RMSE value, there is far more potential that can be realized through the incorporation of other predictors such as "genres" and "releaseYear". A drastic RMSE reduction would also be possible with the use of a larger dataset. The MovieLens 10M dataset could be swapped for the 25M dataset offered on the GroupLens website to nearly triple the number of ratings at my disposal.

While there is clear room for improvement, the Regularized Movie + User Effects Model satisfies the primary goal of the MovieLens Project, making this venture an overall success.

Limitations

My MovieLens Project encounter several limitations that I hope to address in the future, perhaps through the assistance of my peer reviewers.

1) Software/Hardware Limitations: I have attempted several advanced machine learning models that could have reduced the RMSE further, such as k-NN models, but the lackluster processing power and RAM of my laptop has currently made it difficult to do so.
2) Exploratory Limitations: A higher understanding of the MovieLens dataset would have allowed me to fine-tune the models further.

Future Works

As summarized above, my attempts at creating a better model has been hampered by the limitations of my device and my understanding of the MovieLens dataset. However, I intend on trying additional models that show much promise.

1) Installing the recosystem package to utilize Matrix Factorization.
2) Utilize PCA (Principal Component Analysis) to improve visualization of the data.

Acknowledgements

I would like to thank Professor Rafael Irizarry, as well as the countless educators, moderators, and supervisors that have contributed to the HarvardX Professional Data Science Certificate Program. Their collective efforts have provided one of the best remote learning experiences I have had to date, and I am extremely grateful to have studied the R programming language under such accredited curriculum. I also share my gratitude and heartfelt thanks to my fellow peers, whose discussions on the interactive forums have given me invaluable insight throughout the journey. Lastly, I would like to acknowledge my parents, who have never stopped supporting my dreams.