MovieLens Capstone Project

Dana Ghioca Robrecht

2022-06-24

I. Introduction

In October 2006, Netflix opened to the wide public a one-million dollars' worth challenge: create a machine learning algorithm that can improve the company's movie recommendation system by 10% and win the big prize and world recognition! More than 50,000 participants from 186 countries took on this challenge. Three years later, in September 2009, the BellKor Pragmatic Chaos Team won the prize beating the runner-up team by 20 minutes! Ever since, machine learning continued to become more a trend of the present than of the future and even helped define the concept of "algorithmic culture."

Our machine learning challenge is built on the **MovieLens 10M** dataset, a large dataset that includes about 10 million data entries representing movie ratings (5 star scale) for about 10,000 movies and 70,000 users (however, not every user rated every movie). This dataset is available from the GroupLens research lab. Along with movie ratings, movie identification number, and user identification number, the dataset also includes movie titles (which incorporate the year of release), genres, and a timestamp which represents the time and date when the rating was recorded. This dataset was split into the **edx** set (includes 90% randomly selected values from the original Movielens 10M set) to be used for constructing the algorithm and the **validation** set (10% of the original set) to be used only for the final testing of the algorithm.

The goal of this capstone project was to build a machine learning algorithm that predicts movie ratings given by users with good accuracy. Accuracy was measured using RMSE (Residual Mean Squared Error) which is a commonly used measure of the differences between values predicted by a model and the true observed values (in our case, those from the **validation** set at the end of the project). In other words, RMSE is the standard deviation of the residuals (i.e., prediction errors): An RMSE larger than 1 means an error larger than one star rating. The target RMSE for this project was 0.86490 or lower.

I used a modeling approach based on the loss function RMSE calculated on linear models that progressively added more biases (also called effects), followed by regularization of the best performing linear model. This approach has successfully brought the RMSE down from 1.05999 to 0.86430 when using the **validation** set for the final check.

II. Methods

1. Data preparation

I started by loading the required R packages and the dataset from the GroupLens webpage as instructed. The data was then split into two sets, the **edx** set (90% of the original dataset), meant to be used for developing the algorithm, and the **validation** set (10% of the original dataset), meant to strictly be used for testing the final best algorithm. The procedure also ensured that all students completing this project obtained comparable results because we set the seed to 1. Using **semi-join()** we ensured that the edx and validation sets had the same users and movies included.

```r
##########################################################
# 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("https://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)`
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)
```

2. Data exploration and visualization

I first explored the two sets (**edx** and **validation**) obtaining summary statistics with **str()**, **summary()**, **dim()**, and **n_distinct()** functions.

```
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 838983707 838
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Adventure|Sci
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
summary(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
##
##
##
```

```
dim(edx)
```

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

```
n_distinct(edx$movieId)
```

```
## [1] 10677
```

```
n_distinct(edx$userId)
```

```
## [1] 69878
```

```
str(validation)
```

```
## Classes 'data.table' and 'data.frame':   999999 obs. of  6 variables:
##  $ userId   : int  1 1 1 2 2 2 3 3 4 4 ...
##  $ movieId  : num  231 480 586 151 858 ...
##  $ rating   : num  5 5 5 3 2 3 3.5 4.5 5 3 ...
##  $ timestamp: int  838983392 838983653 838984068 868246450 868245645 868245920 1136075494 1133571200 844416936 8
##  $ title    : chr  "Dumb & Dumber (1994)" "Jurassic Park (1993)" "Home Alone (1990)" "Rob Roy (1995)" ...
##  $ genres   : chr  "Comedy" "Action|Adventure|Sci-Fi|Thriller" "Children|Comedy" "Action|Drama|Romance|War" ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
summary(validation)
```

```
##      userId          movieId          rating        timestamp        
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08  
##  1st Qu.:18096   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.467e+08  
##  Median :35768   Median : 1827   Median :4.000   Median :1.035e+09  
##  Mean   :35870   Mean   : 4108   Mean   :3.512   Mean   :1.033e+09  
##  3rd Qu.:53621   3rd Qu.: 3624   3rd Qu.:4.000   3rd Qu.:1.127e+09  
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09  
##     title              genres         
##  Length:999999      Length:999999     
##  Class :character   Class :character  
##  Mode  :character   Mode  :character  
## 
## 
## 
```

```
dim(validation)
```

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

```
n_distinct(validation$movieId)
```

```
## [1] 9809
```

```
n_distinct(validation$userId)
```

```
## [1] 68534
```

There are six variables (i.e., columns) in each dataset: userId, movieId, rating, timestamps, title, and genres. The last two are character variables, and the rest are integer or numerical variables. I also learned that **edx** has 9,000,055 entries (i.e., rows) and is made up of 10,677 distinct movies and 69,878 distinct users. The **validation** set has 999,999 entries, 9,809 distinct movies, and 69,532 distinct users. The mean rating was the same for both sets (3.512) as was the median (4), and the ratings varied between 0.5 and 5 stars.

I noticed that timestamp (the date and time a movie was reviewed by a user) is not very helpful in the current format, so I extracted the date and rounded it to week units. I have done this on both the **edx** and **validation** sets and then checked that the same number of dates are included in both sets.

```
library(lubridate)
edx <- edx %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))

validation <- validation  %>%
  mutate(date = round_date(as_datetime(timestamp), unit = "week"))

validation <- validation %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
```

I also noticed that the year when a movie was released is included in the title, so I extracted the release year and created a new column named "released" in the **edx** and **validation** sets.

```r
released <- as.numeric(str_sub(edx$title, start = -5, end = -2))
edx <- edx %>% mutate(released = released)
head(edx)
```

```
##    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       date released
## 1:                Comedy|Romance 1996-08-04     1992
## 2:         Action|Crime|Thriller 1996-08-04     1995
## 3:  Action|Drama|Sci-Fi|Thriller 1996-08-04     1995
## 4:       Action|Adventure|Sci-Fi 1996-08-04     1994
## 5: Action|Adventure|Drama|Sci-Fi 1996-08-04     1994
## 6:       Children|Comedy|Fantasy 1996-08-04     1994
```

```r
dim(edx)
```

```
## [1] 9000055       8
```

```r
released <- as.numeric(str_sub(validation$title, start = -5, end = -2))
validation <- validation %>% mutate(released = released)
head(validation)
```

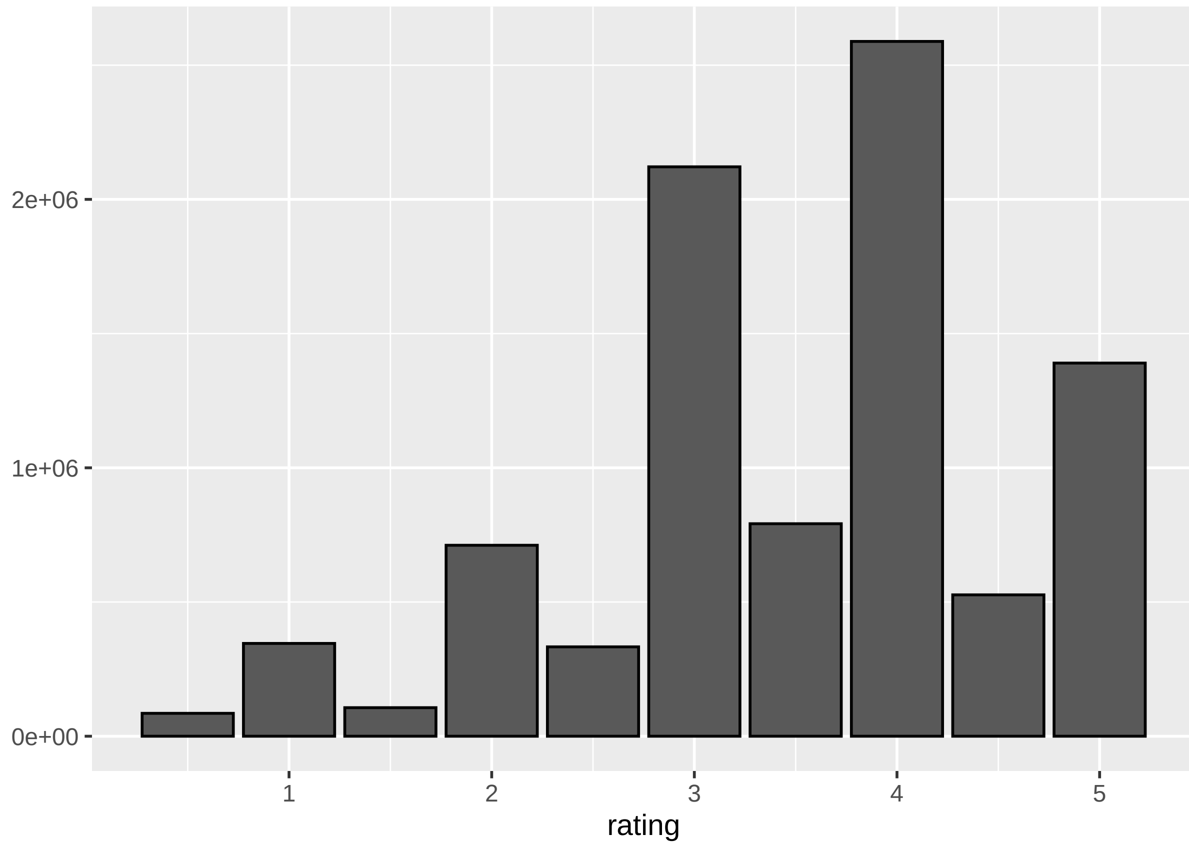
```
##    userId movieId rating timestamp
## 1:      1     231      5 838983392
## 2:      1     480      5 838983653
## 3:      1     586      5 838984068
## 4:      2     151      3 868246450
## 5:      2     858      2 868245645
## 6:      2    1544      3 868245920
##                                                  title
## 1:                                  Dumb & Dumber (1994)
## 2:                                  Jurassic Park (1993)
## 3:                                     Home Alone (1990)
## 4:                                        Rob Roy (1995)
## 5:                                 Godfather, The (1972)
## 6: Lost World: Jurassic Park, The (Jurassic Park 2) (1997)
##                                    genres       date released
## 1:                                 Comedy 1996-08-04     1994
## 2:        Action|Adventure|Sci-Fi|Thriller 1996-08-04     1993
## 3:                        Children|Comedy 1996-08-04     1990
## 4:                Action|Drama|Romance|War 1997-07-06     1995
## 5:                            Crime|Drama 1997-07-06     1972
## 6: Action|Adventure|Horror|Sci-Fi|Thriller 1997-07-06     1997
```

```r
dim(validation)
```

```
## [1] 999999      8
```

From this point on I only worked with the **edx** set for visualization and building the predictive model until the very last step of checking the final model with the **validation** set. To visualize the distribution of ratings and given that the ratings are not really continuous data, but rather categorical data (i.e., there are 10 categories), a bar plot was my choice for visualizing the rating distribution instead of a histogram (appropriate for continuous variables).

```
qplot(rating, data = edx, color = I("black"), geom = "bar")
```



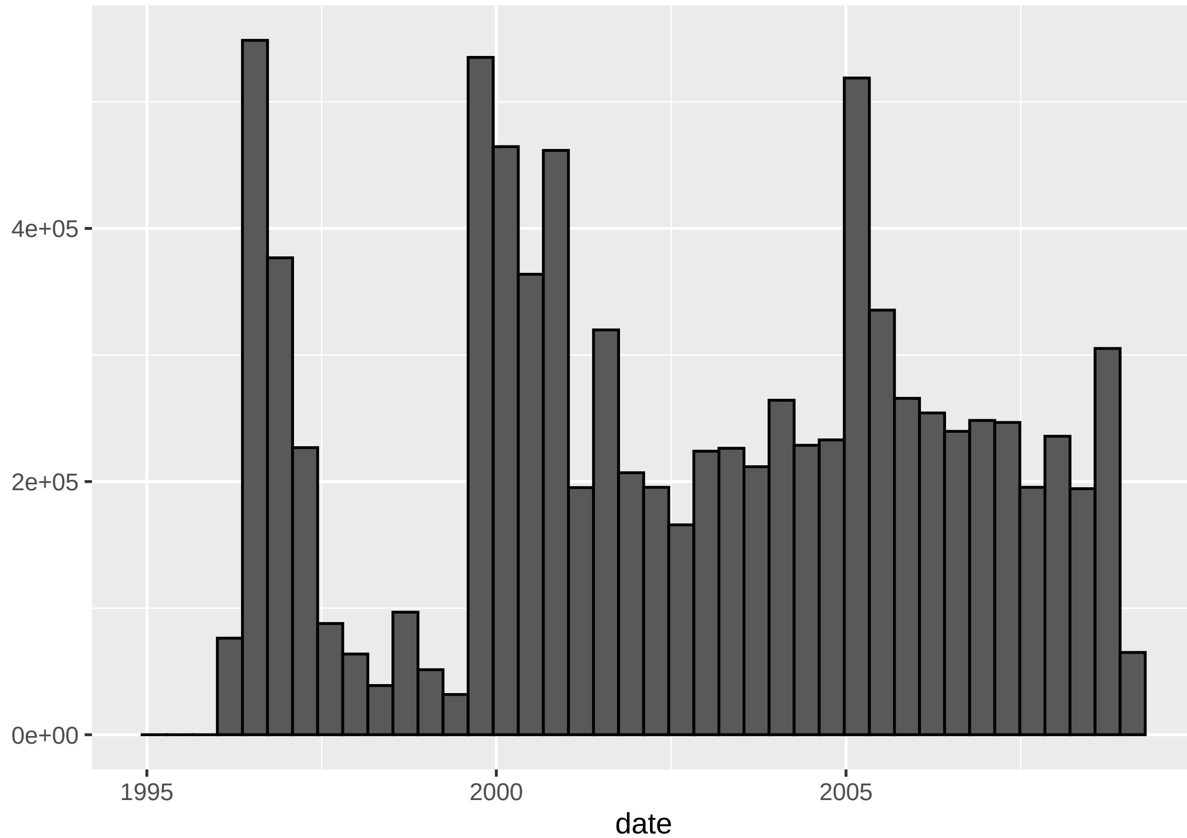
The most frequent rating was 4, followed by 3, and then 5, indicating a possible bias towards higher ratings.

I also looked at the distribution of other variables in the dataset as these were the potential effects to account for in my models.

```
range(edx$date)
```

```
## [1] "1995-01-08 UTC" "2009-01-04 UTC"
```

```
qplot(date, data = edx, bins = 40, color = I("black"))
```

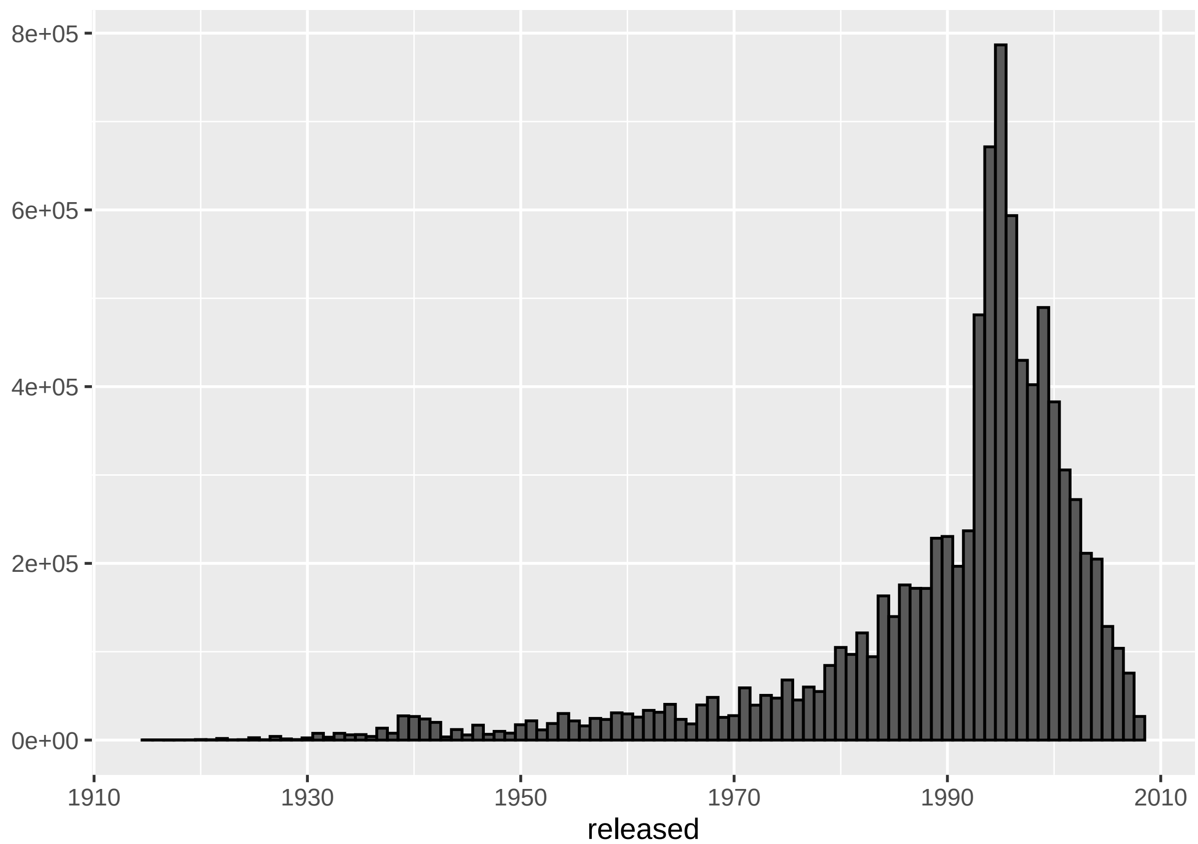


The dates of review ranged from 1995 to 2009 and had an interesting distribution showing that there was a review peak around 1996, then another surge in reviews around 2000-2001, and then again one in 2005. (One may wonder, is there a five-year review resurgence pattern?)

```
range(edx$released)
```

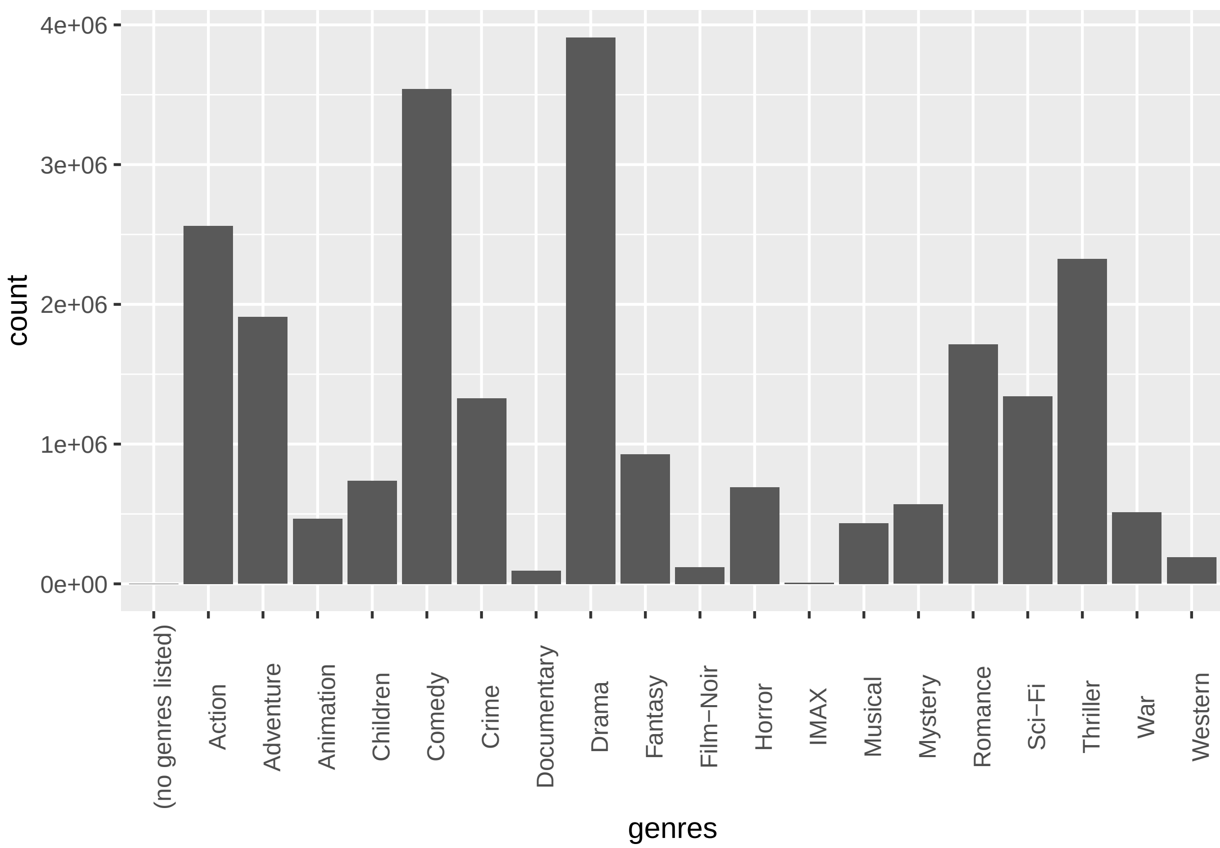
```
## [1] 1915 2008
```

```
qplot(released, data = edx, binwidth = 1, color = I("black"))
```



Movies were released between 1915 and 2008, but the distribution is skewed to the left with a peak in the mid to late 90's suggesting these were the most popular movies.

```
edx %>% separate_rows(genres, sep = "\\|") %>%
  ggplot(aes(genres)) +
  geom_bar() +
  theme(axis.text.x = element_text(angle=90))
```



After separating into individual genres the original genre category which includes multiple designations for each movie, the most commonly rated movie genres were Drama, Comedy, and Action whereas the least frequently rated were IMAX, Documentary, and Film-Noir.

3. Building the model

My approach for building the algorithm was inspired by the method described in Course 8 of this certification program in the "Recommendation systems" chapter, using linear regression models followed by regularization. However, to reach the target RMSE, I added several more effects and also expanded the regularization step. I started by partitioning the **edx** set in training and test sets, similar to how we split the **MovieLens** set, but using 20% of the **edx** dataset for testing and 80% of it for training.

The simplest model (the naïve model) assumes that all movies and users produce the same rating. The observed variability in the ratings in the data set is then due only to random variation. I calculated the "true" rating mean (I called it "Basic Average") by calculating the average of all ratings in the **edx** training data set. The RMSE of this basic model was 1.0599.

To this basic model I then added successively various effects and checked how the RMSE improved. For the first model, I added the movie effect, then in the second model I added the user effect to the previous model. Next model also included the genre effect, and then I added the date of movie review followed lastly by adding the year of release. The RMSE progresively improved, but it still did not reach the target RMSE of 0.86490. The next and final step was to perform regularization on this model that included all five effects. I used cross-validation to find the lambda that minimized the RMSE and then found the RMSE for the model ran with this optimal lambda. The RMSE was adequate, and thus for the final check, I used the **validation** set and also obtained a satisfactory RMSE.

I created a table to keep track of the modeling results.

III. Results and Discussion

I first partitioned the **edx** set in a training (80% of the **edx** set) and test set (20% of the **edx** set).

```
set.seed(1, sample.kind="Rounding")
index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
edx_train <- edx[-index,]
edx_temp <- edx[index,]


# Make sure userId and movieId in test set are also in train set
edx_test <- edx_temp %>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")

# Add rows removed from test set back into train set
removed <- anti_join(edx_temp, edx_test)
edx_train <- rbind(edx_train, removed)

rm(index, edx_temp, removed)
```

The simplest model assumes that all movies and users produce the same rating and observed variability is due only to random variation. I calculated the least square estimate (LSE) for this "true" rating mean by calculating the average of the ratings in the **edx** training set.

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

```
## [1] 3.512478
```

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

```
## [1] 1.059904
```

The naive RMSE was 1.059904. I created a table to keep track of the model improvements.

```
rmse_results <- tibble(method ="Basic Average", RMSE = naive_rmse)
rmse_results
```

```
## # A tibble: 1 x 2
##   method         RMSE
##   <chr>         <dbl>
## 1 Basic Average  1.06
```

In the next step I added in the model a movie bias. I added the new RMSE to the table.

```
movie_avgs <- edx_train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

predicted_ratings_1 <- mu + edx_test %>%
  left_join(movie_avgs, by = "movieId")  %>%
  .$b_i
```

```r
model_1_rmse <- RMSE(predicted_ratings_1, edx_test$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model 1 - Movie Effect Model",
                                     RMSE = model_1_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Basic Average	1.0599043
Model 1 - Movie Effect Model	0.9437429

Adding a movie bias factor in the model decreased the RMSE to 0.94374. Next, I added in the model a user bias.

```r
user_avgs <- edx_train  %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

predicted_ratings_2 <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred

model_2_rmse <- RMSE(predicted_ratings_2, edx_test$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model 2 - Movie + User Effects Model",
                                     RMSE = model_2_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Basic Average	1.0599043
Model 1 - Movie Effect Model	0.9437429
Model 2 - Movie + User Effects Model	0.8659319

Adding a user bias in the model further decreased the RMSE to 0.86593. For the next model, I added a genre bias.

```r
genre_avgs <- edx_train  %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId')%>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))

genre_avgs %>% separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(count = n()) %>%
  arrange(desc(count))
```

```
## # A tibble: 20 x 2
##    genres             count
##    <chr>              <int>
```

```
##  1 Drama                308
##  2 Comedy               270
##  3 Action               252
##  4 Adventure            249
##  5 Thriller             223
##  6 Fantasy              198
##  7 Romance              176
##  8 Sci-Fi               172
##  9 Crime                166
## 10 Mystery              140
## 11 Horror               134
## 12 Children             122
## 13 Animation            113
## 14 Musical               93
## 15 War                   85
## 16 Western               69
## 17 Film-Noir             39
## 18 Documentary           28
## 19 IMAX                  12
## 20 (no genres listed)     1
```

```
predicted_ratings_3 <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  .$pred

model_3_rmse <- RMSE(predicted_ratings_3, edx_test$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model 3 - Movie + User + Genre Effects Model",
                                     RMSE = model_3_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Basic Average	1.0599043
Model 1 - Movie Effect Model	0.9437429
Model 2 - Movie + User Effects Model	0.8659319
Model 3 - Movie + User + Genre Effects Model	0.8655941

This model with three factors had an RMSE of 0.86559, better than the previous one. Next, I added the date of review effect to the previous model.

```
date_avgs <- edx_train %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  group_by(date) %>%
  summarize(b_d = mean(rating - mu - b_i - b_u - b_g))

predicted_ratings_4 <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(date_avgs, by='date') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_d) %>%
```

```
  .$pred

model_4_rmse <- RMSE(predicted_ratings_4, edx_test$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model 4 - Movie + User + Genre + Date Effects Model",
                                     RMSE = model_4_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Basic Average	1.0599043
Model 1 - Movie Effect Model	0.9437429
Model 2 - Movie + User Effects Model	0.8659319
Model 3 - Movie + User + Genre Effects Model	0.8655941
Model 4 - Movie + User + Genre + Date Effects Model	0.8654875

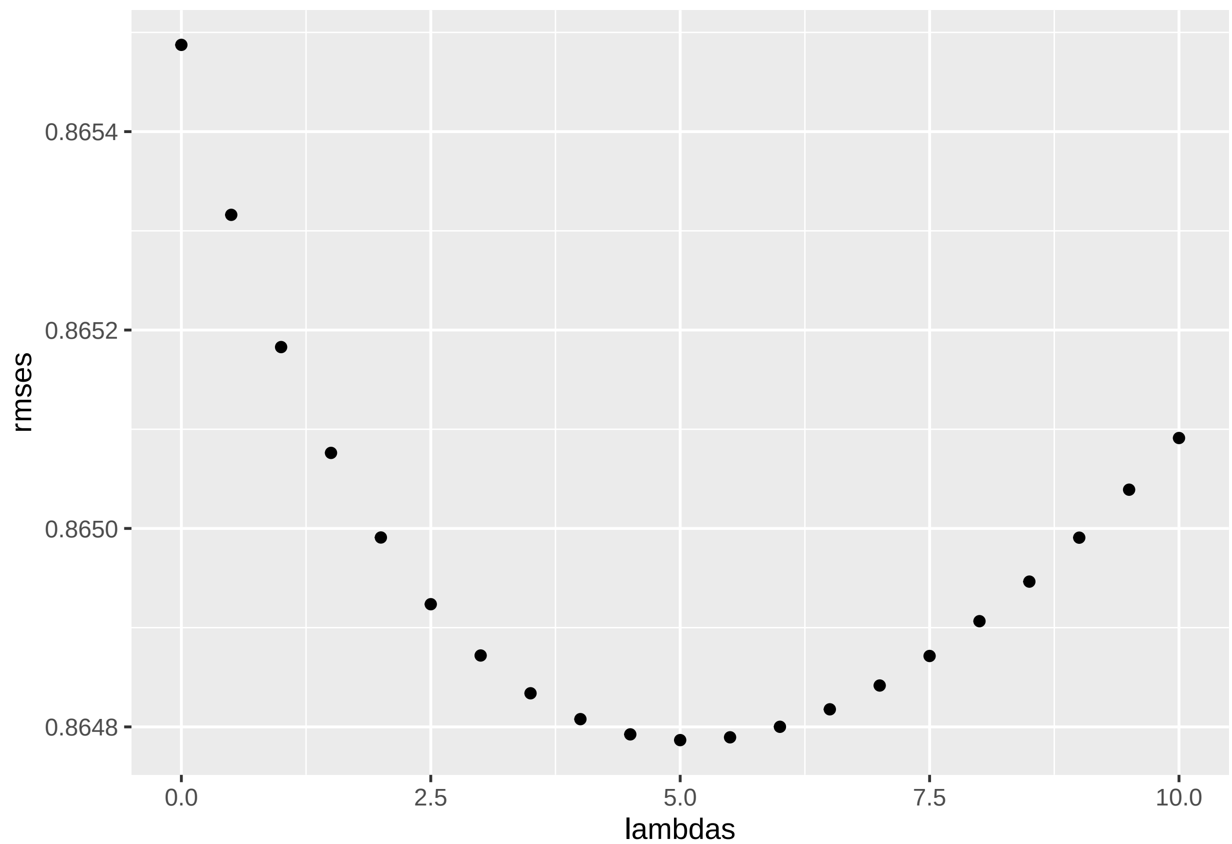
Adding the date of review modestly reduced the RMSE of the model to 0.86549. Thus, I added lastly the year of release to the model.

```
release_avgs <- edx_train %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(date_avgs, by='date') %>%
  group_by(released) %>%
  summarize(b_r = mean(rating - mu - b_i - b_u - b_g - b_d))

predicted_ratings_5 <- edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(date_avgs, by='date') %>%
  left_join(release_avgs, by ="released") %>%
  mutate(pred = mu + b_i + b_u + b_g + b_d + b_r) %>%
  .$pred

model_5_rmse <- RMSE(predicted_ratings_5, edx_test$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model 5 - Movie + User + Genre + Date + Release Effects Model",
                                     RMSE = model_5_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Basic Average	1.0599043
Model 1 - Movie Effect Model	0.9437429
Model 2 - Movie + User Effects Model	0.8659319
Model 3 - Movie + User + Genre Effects Model	0.8655941
Model 4 - Movie + User + Genre + Date Effects Model	0.8654875
Model 5 - Movie + User + Genre + Date + Release Effects Model	0.8652550

So far this was the best model with an RMSE of 0.86526, but still this was higher than the target of 0.86490. Thus, I performed regularization on this best model. I used cross-validation to find the lambda that minimized the RMSE.

```r
lambdas <- seq(0, 10, 0.5)

rmses <- sapply(lambdas, function(l){

      mu <- mean(edx_train$rating)

      b_i_r <- edx_train %>%
      group_by(movieId) %>%
      summarize(b_i_r = sum(rating - mu)/(n()+l))

      b_u_r <- edx_train %>%
      left_join(b_i_r, by="movieId") %>%
      group_by(userId) %>%
      summarize(b_u_r = sum(rating - b_i_r - mu)/(n()+l))

      b_g_r <- edx_train %>%
        left_join(b_i_r, by="movieId") %>%
        left_join(b_u_r, by="userId") %>%
        group_by(genres) %>%
        summarize(b_g_r = sum(rating - b_i_r - b_u_r - mu)/(n()+l))

      b_d_r <- edx_train %>%
        left_join(b_i_r, by="movieId") %>%
        left_join(b_u_r, by="userId") %>%
        left_join(b_g_r, by="genres") %>%
        group_by(date) %>%
        summarize(b_d_r = sum(rating - b_i_r - b_u_r - b_g_r - mu)/(n()+l))

      b_r_r <- edx_train %>%
        left_join(b_i_r, by="movieId") %>%
        left_join(b_u_r, by="userId") %>%
        left_join(b_g_r, by="genres") %>%
        left_join(b_d_r, by='date') %>%
        group_by(released) %>%
        summarize(b_r_r = mean(rating -b_i_r - b_u_r - b_g_r - b_d_r - mu)/(n()+l))

      predicted_ratings_6 <- edx_test %>%
      left_join(b_i_r, by = "movieId") %>%
      left_join(b_u_r, by = "userId") %>%
      left_join(b_g_r, by ="genres") %>%
      left_join(b_d_r, by ="date") %>%
      left_join(b_r_r, by ="released") %>%
      mutate(pred = mu + b_i_r + b_u_r + b_g_r + b_d_r + b_r_r) %>%
      .$pred

       return(RMSE(predicted_ratings_6, edx_test$rating))
       })

qplot(lambdas, rmses)
```




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

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

```
model_6_rmse <- min(rmses)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model 6 - Regularized Movie + User + Gender + Date + Release Effects M
                                     RMSE = model_6_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Basic Average	1.0599043
Model 1 - Movie Effect Model	0.9437429
Model 2 - Movie + User Effects Model	0.8659319
Model 3 - Movie + User + Genre Effects Model	0.8655941
Model 4 - Movie + User + Genre + Date Effects Model	0.8654875
Model 5 - Movie + User + Genre + Date + Release Effects Model	0.8652550
Model 6 - Regularized Movie + User + Gender + Date + Release Effects Model	0.8647867

The optimal lambda was 5.0 and the RMSE of the regularized model with this lambda was 0.86479, which was below the target of 0.86490!

For the final check, I used the **validation** set to calculate the predicted ratings which I then compared to the actual ratings in this **validation** set to obtain the validation RMSE.

```r
l <- lambda
  mu_edx <- mean(edx$rating)
  b_i_r <- edx %>%
    group_by(movieId) %>%
    summarize(b_i_r = sum(rating - mu_edx)/(n()+l))

  b_u_r <- edx %>%
    left_join(b_i_r, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u_r = sum(rating - b_i_r - mu_edx)/(n()+l))

  b_g_r <- edx %>%
    left_join(b_i_r, by="movieId") %>%
    left_join(b_u_r, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g_r = sum(rating - b_i_r - b_u_r - mu_edx)/(n()+l))

  b_d_r <- edx %>%
    left_join(b_i_r, by="movieId") %>%
    left_join(b_u_r, by="userId") %>%
    left_join(b_g_r, by="genres") %>%
    group_by(date) %>%
    summarize(b_d_r = sum(rating - b_i_r - b_u_r - b_g_r - mu_edx)/(n()+l))

  b_r_r <- edx %>%
    left_join(b_i_r, by="movieId") %>%
    left_join(b_u_r, by="userId") %>%
    left_join(b_g_r, by="genres") %>%
    left_join(b_d_r, by='date') %>%
    group_by(released) %>%
    summarize(b_r_r = mean(rating -b_i_r - b_u_r - b_g_r - b_d_r - mu_edx)/(n()+l))

  predicted_ratings_final <- validation %>%
    left_join(b_i_r, by = "movieId") %>%
    left_join(b_u_r, by = "userId") %>%
    left_join(b_g_r, by ="genres") %>%
    left_join(b_d_r, by ="date") %>%
    left_join(b_r_r, by ="released") %>%
    mutate(pred = mu_edx + b_i_r + b_u_r + b_g_r + b_d_r + b_r_r) %>%
    .$pred

  final_rmse_check <- RMSE(predicted_ratings_final, validation$rating)
  final_rmse_check
```

```
## [1] 0.8643062
```

The RMSE of the regularized model using the **validation** set was **0.86431**, which is below the 0.86490 threshold, thus I successfully completed the challenge!

IV. Conclusions

The final model, which contained five effects and used regularization to improve the accuracy of the linear model, achieved an RMSE of .86431. Further improvements on this method could include using different lambdas for each effect in the model. Other improvements could refer to the type of effects included in the model as well as different modeling methods. “Baseline” model modification, such as those described by the

BellKor winning team, could include adding temporal effects. One would modify the movie effect to account for the popularity of a movie rating changing over time due to, for example, the decrease in popularity of an actor. Another change would modify a user effect to reflect a temporal shift in their ratings system as, for example, users may give more favorable rating over time. The frequency with each a user gives ratings in a day can also be accounted for in such complex models.

There are also other approaches such as matrix factorization using the recosystem package, neighborhood models, ensemble of tree models etc. that can improve the accuracy of the model. However, these and other advanced methods, require significant computational power which can be a limiting factor for some students.

Given the interest in recommendation systems due to the many streaming services that exist today, predictive algorithms will continue to be developed and it will be exciting to see how much the rating predictions can be improved.

V. References

Irizarry, R. A. 2019. Introduction to Data Science: Data Analysis and Prediction Algorithms in R. Chapman and Hall/CRC. 719 pages

Koren, Y. 2009. The BellKor Solution to the Netflix Grand Prize. https://www2.seas.gwu.edu/~simhaweb/champalg/cf/papers/KorenBellKor2009.pdf

Hallinan, B. and Striphas, T. 2016 Recommended for you: The Netflix Prize and the production of algorithmic culture. New Media & Society, 18(1), pp. 117–137. doi: 10.1177/1461444814538646.

https://grouplens.org/datasets/movielens/10m/

https://rmarkdown.rstudio.com/