MovieLens Capstone Project

Neill Smith

2022-09-06

Introduction

For most vendors of goods, services, or content, the primary goal is to provide a product that consumers want. If a vendor has a staggering number of options how can they ensure that a customer can find the option that best fits their needs? Customers may stop returning if they run out of time or patience before finding an option that strikes their fancy. Businesses with large selections can help customers sift through what they have to offer by using a recommendation system. This system would need to be as accurate as possible to keep customers engaged; bad recommendations could be even worse than no recommendations. Enter machine learning. By taking the data gathered on the products and how past customers have rated them we can try to predict which products a costumer will respond to best.

The data set used is a subset of the MovieLens data set from GroupLens research lab at the University of Minnesota. The data set of 10 million ratings is divided into a training set (edx) of 90% of the data and a validation set (validation) of 10% of the data. The goal is to create an algorithm that predicts the rating a user will give a film. This could then be converted into a recommendation engine by ranking predicted ratings by user. To evaluate this data we will use a root mean squared error (RMSE) loss function. Factors will be identified in the data that can be used to reduce the RMSE of our predictions and applied to achieve our target RMSE of 0.8649000 or below with the validation set. We will use cross-validation to adjust tuning parameters and to monitor the improvements in our model as factors are identified and implemented.

Methodology & Analysis

This report will examine the MovieLens data set and iterate through several models which attempt to predict the rating a user will give a movie given characteristics of that movie. Aspects of the data examined and used in the model include:

1. The movie's average rating across all users.
2. The average rating a user has given other films.
3. The average rating given to movies in the same genre.
4. How much time has passed since the movie first released.

Models were built using the training data starting with the simple mean, progressing through linear factors, applying a regularization factor, and finally comparing against the validation to arrive at the project's final RMSE.

Data Retrieval and Cleaning

This section will cover the code provided by the course to generate the edx set and validation set from the source MovieLens 10M data set. It checks for required packages and installs them if necessary, splits the

data into 90% test set (edx) and 10% validation set (validation), and then removes the variables that will not be used later.

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

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

# MovieLens 10M data set:
# 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)
```

Setup Cross-Validation

Here we further divide the edx set into a test set and a train set using the same methodology as above so we can employ cross-validation for our model. The train set will be 90% of the edx set and the test set will be the remaining 10%. The data is further analyzed to ensure movieIds and userIds in the test set are also in the train set. Finally we remove the variables that will not be used later.

```
# test set will be 10% of edx data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train <- edx[-test_index,]
temp <- edx[test_index,]

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

# Add rows removed from test set back into edx set
removed <- anti_join(temp, test)
train <- rbind(train, removed)

# Remove extra variables
rm(test_index, temp, removed)
```

Data Exploration

Basic Data Structure

First we begin with some explorations of the underlying data. Let us look at the structure of the data we have setup:

```
## 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 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ac
##  - attr(*, ".internal.selfref")=<externalptr>
```

We can see the data is organized into a table with 6 variables:

1. userID - A unique identifier of a user stored as an integer.

2. movieID - A unique identifier of a movie stored as a number.
3. rating - The rating give by the user (userID) for the film (movieID) stored as a number.
4. timestamp - An integer for when the rating was submitted which appears to be a unicode timestamp.
5. title - The title of the movie, including the year of release after the title surrounded by parenthesis, stored as a character string.
6. genres - The genre or combination of genres the film is categorized in. Multiple genres are delimited by '|'s. Stored as a character string

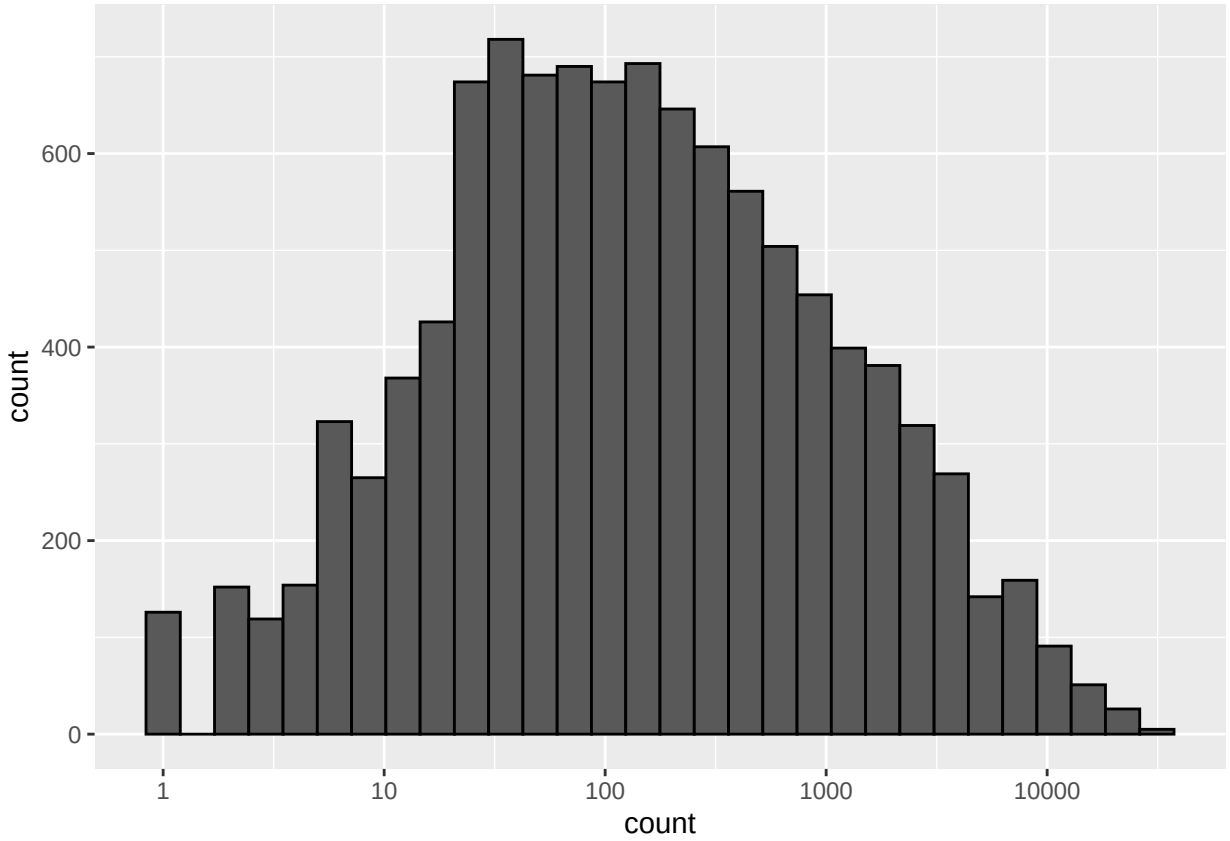
A few of these fields will need some additional wrangling to be useful but the data is largely clean as is. Let us explore a few fields in more depth.

Popularity

We first look at the most reviewed movies to get an idea for the upper bound of the number of ratings per movie. This table ranks movieIds by number of ratings including the title to be more readable:

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                  count
##      <dbl> <chr>                                                  <int>
##  1     296 Pulp Fiction (1994)                                    31362
##  2     356 Forrest Gump (1994)                                    31079
##  3     593 Silence of the Lambs, The (1991)                       30382
##  4     480 Jurassic Park (1993)                                   29360
##  5     318 Shawshank Redemption, The (1994)                       28015
##  6     110 Braveheart (1995)                                      26212
##  7     457 Fugitive, The (1993)                                   25998
##  8     589 Terminator 2: Judgment Day (1991)                      25984
##  9     260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25672
## 10     150 Apollo 13 (1995)                                       24284
## # ... with 10,667 more rows
```

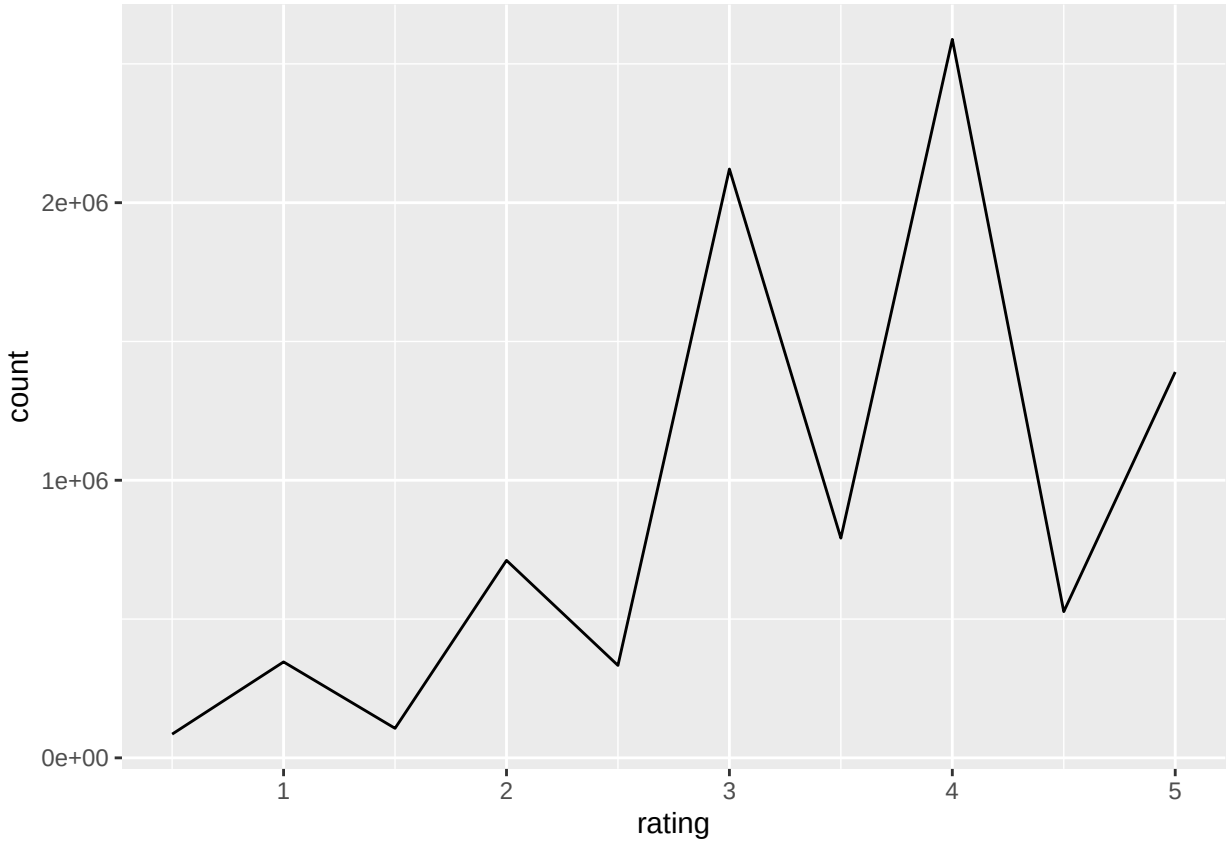
We can see that some titles have tens of thousands of ratings here, but what does the full distribution look like? We will chart the number of ratings per title:



Here we use a log base 10 scale because while some films have tens of thousands of ratings we can see the distribution peaks around 100.

Preference for Full Star Ratings

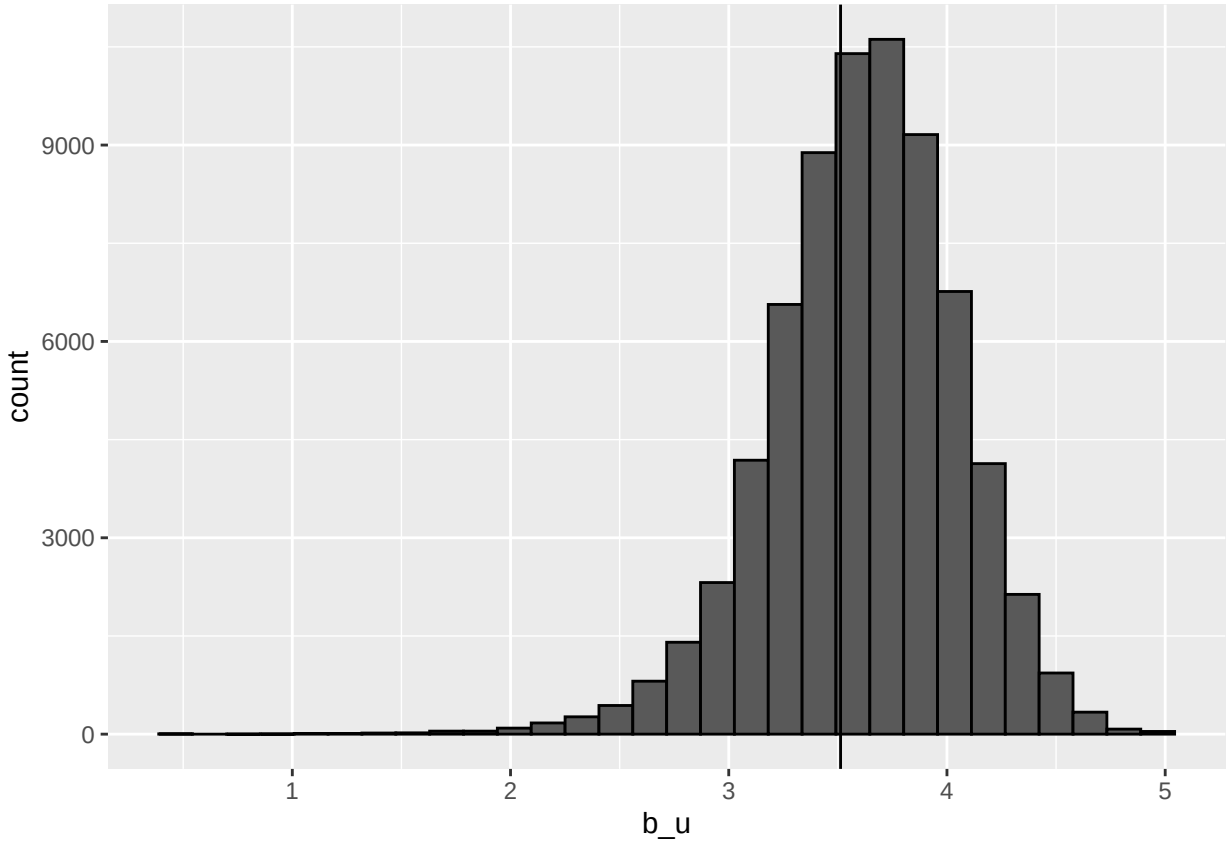
We will chart the number of ratings given for each rating option:



In this plot we can see there are clear dips in the ratings which are being given. With no preference we would expect to see a smooth curve, but there are dips for half star ratings indicating a preference for full star ratings over half star ratings.

Users

Next we will examine how ratings vary between users. We start by charting the average ratings by user.



The vertical line in the chart is showing the average rating, and we can see that the average rating by user is higher than the average rating overall. This suggests there are a small number of users with a lot of ratings who tend to be harder on films than the average user. We should take these user effects into account in our model.

Genre Makeup

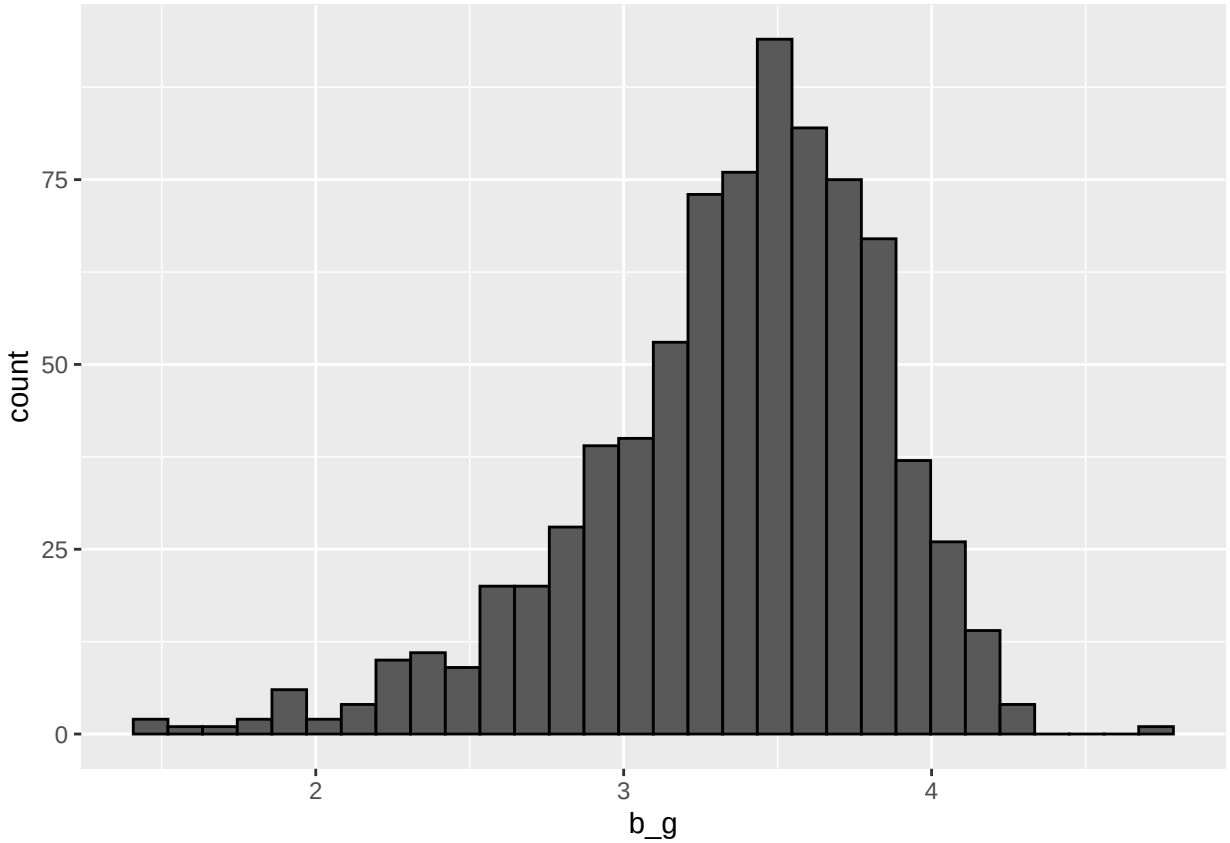
Now we will take a closer look at the genre makeup of the films in the dataset. Here we chart the counts by genre:

```
# create a list of singular genres extracted from the genre field
genres <- unique((edx %>% filter(!str_detect(edx$genres, "\\|")))$genres)
# count the number of films by genre
sapply(genres, function(g) {
  sum(str_detect(edx$genres, g))
}) %>% knitr::kable()
```

	x
Comedy	3540930
Drama	3910127
Thriller	2325899
Western	189394
Horror	691485
Documentary	93066

	x
Action	2560545
Romance	1712100
Sci-Fi	1341183
Children	737994
Adventure	1908892
Animation	467168
Musical	433080
Film-Noir	118541
Crime	1327715
War	511147
Mystery	568332
Fantasy	925637
IMAX	8181
(no genres listed)	7

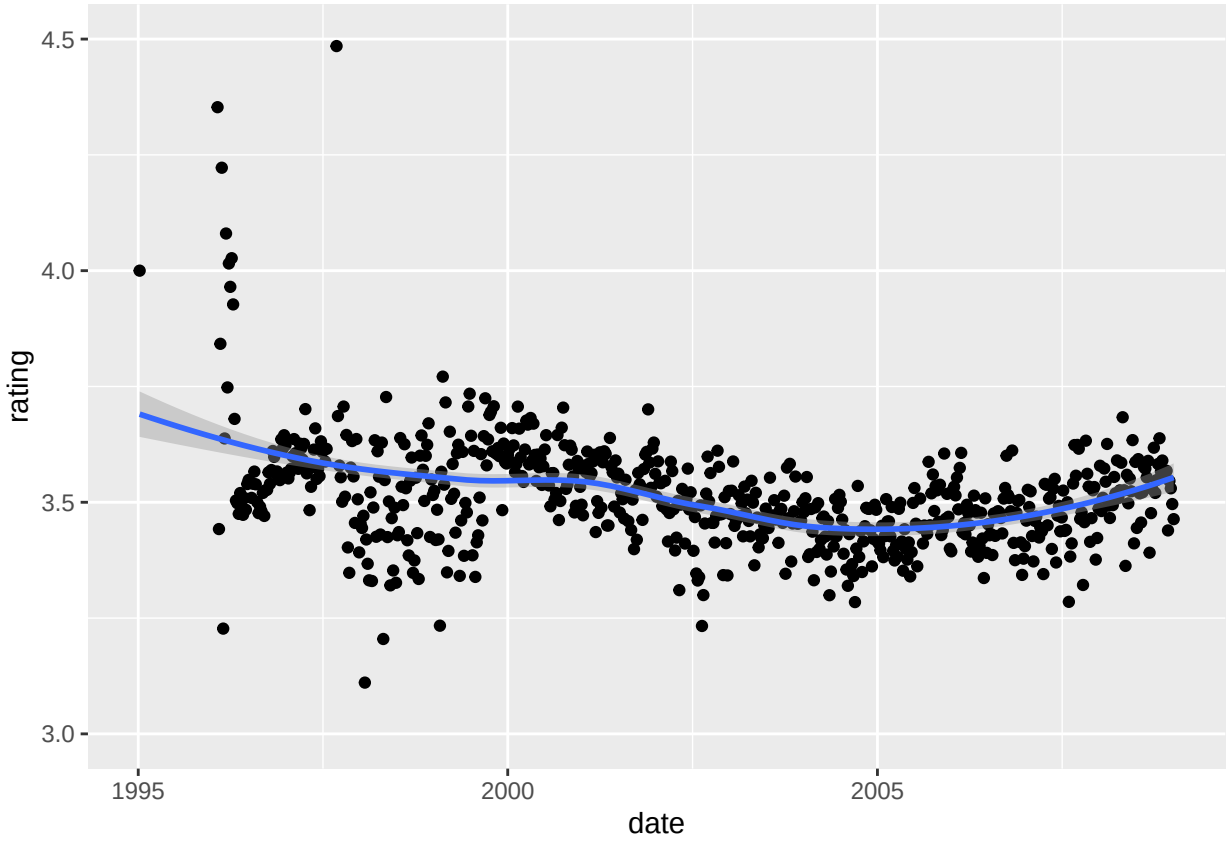
We can see some genres are better represented than others, and we can infer from the counts in the table that most films belong to more than one genre. Is there a relationship between genres and ratings? Let us find out:



The distribution centers around the overall average rating pretty well, but there is clearly a genre effect in action here. In addition the tail on the left side towards 0 stars is much longer than the tail on the right side. We will attempt to take genre into account in our model.

Time Preferences

Taking a look at the timestamp values, we can perform some data wrangling to group when ratings submissions took place and see if there is a time trend in the ratings i.e. are people getting harder on films or easier on them over time?



There seems to be a small effect over time, but perhaps something else is going on here. Let us dig a little deeper into the relationship between the submission dates and the year of release.

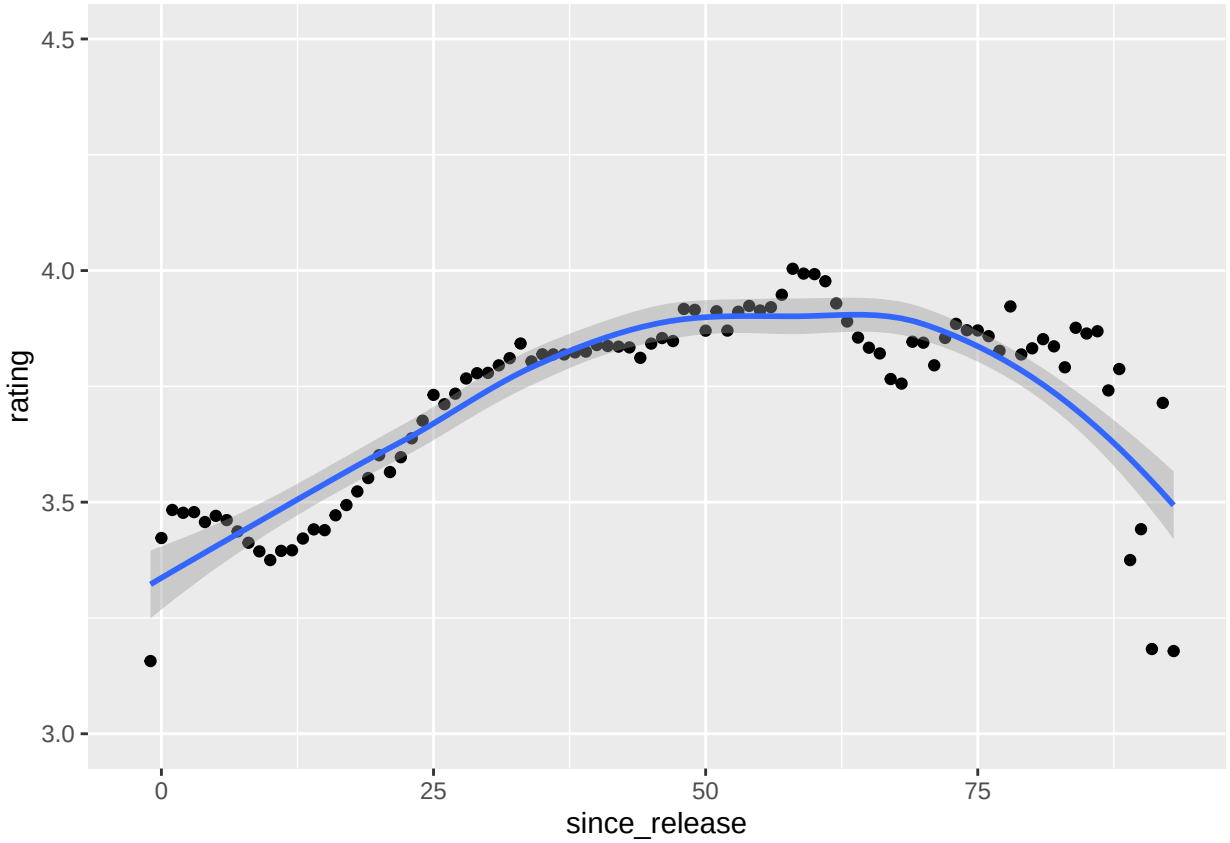
We take some additional data wrangling steps to pull the release years out of the titles by searching for 4-digit strings between a "(" and a ")" at the end of the string. First we will check that our data wrangling logic is pulling out 4-digit years by examining the unique values our logic produces.

```
# check the results of our string match are returning 4 digit years
unique(mutate(edx,
              release_year = as.numeric(str_match(title,"\\((\\d{4})\\)$")[,2])
              )$release_year)
```

```
##  [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
```

These appear to be valid 4-digit years so we are safe to proceed.

Let us examine how the average rating changes with time since the release year of the film. We will plot the average rating by number of years since release. To better compare to the previous chart we will use the same y-axis scale.



There does appear to be a relationship here; generally the older a movie is when it is rated the better the rating. A couple of other interesting results pop out as well. When a film is older than ~60 years, the improvement diminishes. For very new films there is also a slight bump above the local average. We will incorporate the time difference between release and rating submission into our model.

Curiously we also saw a handful of ratings that were made *before* the film was officially released, shown as less than 0 on the chart. Let us confirm our methodology is accurate by examining these entries further:

```
edx %>%
    mutate(date = as_datetime(timestamp),
           release_year = as.numeric(str_match(title,"\\((\\d{4})\\)$")[,2]),
           since_release = year(as_datetime(timestamp))-
             as.numeric(str_match(title,"\\((\\d{4})\\)$")[,2])) %>%
  filter (since_release < 0)
```

```
##       userId movieId rating  timestamp                  title
##    1:    785     981    3.0  844464462 Dangerous Ground (1997)
##    2:   1468     879    2.0  841308568       Relic, The (1997)
##    3:   1583     981    1.0  842861387 Dangerous Ground (1997)
##    4:   1766     870    5.0  839441876     Gone Fishin' (1997)
##    5:   1766     879    5.0  840812687       Relic, The (1997)
##   ---
## 171:  70508     981    2.0  849375725 Dangerous Ground (1997)
```

```
## 172:  70905   38188    2.5 1129236336           Bubble (2006)
## 173:  71311     987    4.0  850130250            Bliss (1997)
## 174:  71391    1355    4.0  849670026       Nightwatch (1997)
## 175:  71518    1355    3.0  849522274       Nightwatch (1997)
##                   genres                date release_year since_release
##   1:              Drama 1996-10-04 21:27:42         1997            -1
##   2:     Horror|Thriller 1996-08-29 08:49:28         1997            -1
##   3:              Drama 1996-09-16 08:09:47         1997            -1
##   4:             Comedy 1996-08-07 18:17:56         1997            -1
##   5:     Horror|Thriller 1996-08-23 15:04:47         1997            -1
##  ---
## 171:              Drama 1996-11-30 17:42:05         1997            -1
## 172: Crime|Drama|Mystery 2005-10-13 20:45:36         2006            -1
## 173:       Drama|Romance 1996-12-09 11:17:30         1997            -1
## 174:     Horror|Thriller 1996-12-04 03:27:06         1997            -1
## 175:     Horror|Thriller 1996-12-02 10:24:34         1997            -1
```

These appear to be real instances where the date of the rating preceds the release date, perhaps from pre-screening or leaving a rating on the site without having seen the film before it premiers. This seems to be extremely rare though; there are only 175 observations in the edx set of over 9 million observations.

We should be safe to proceed with this as a factor for our model so we will add this column to edx, test, and train for later use.

```
edx <- edx %>%
  mutate(since_release = year(as_datetime(timestamp))-
           as.numeric(str_match(title,"\\((\\d{4})\\)$")[,2]))
train <- train %>%
  mutate(since_release = year(as_datetime(timestamp))-
           as.numeric(str_match(title,"\\((\\d{4})\\)$")[,2]))
test <- test %>%
  mutate(since_release = year(as_datetime(timestamp))-
           as.numeric(str_match(title,"\\((\\d{4})\\)$")[,2]))
```

Modeling

We will now walk though each of the iterations of our model from a very basic guess through the final form. With the addition of each new factor we will use a loss function to see if we have improved and by how much. For this model the loss function chosen was Root Mean Squared Error (RMSE) which takes the square root of the average squared difference between the true ratings and the predicted ratings. To calculate RMSE, we will create the following function and save it to RMSE for repeat use:

```
# create RMSE funciton to evaluate model results
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Simple Average

We begin by finding the average movie rating in the train data set and use that to predict all ratings. We will call this model "Just the average" because that is all we are doing for this result. Any model with additional factors should be able to improve on this one so we will use this as a baseline.

```
# find average rating in train set
mu_hat <- mean(train$rating)
# Calculate RMSE with simple average as guess
naive_rmse <- RMSE(test$rating, mu_hat)
# add naive_rmse to a table of rmse results
rmse_results <- tibble(method = "Just the average", RMSE = naive_rmse)
```

The resulting RMSE is added to our table of results for later comparison:

method	RMSE
Just the average	1.060054

Movie Effect

Next we look at the effect of other ratings for the same movie. We take each film and calculate the average difference between its rating and the mean rating of the overall dataset to arrive at a movie effect factor which is saved in movie_avgs as b_i.

```
# calculate a movie factor,
# average difference between a movies score and the overall average score
movie_avgs <- train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))
# prediction based on per-movie average combined with overall average
predicted_ratings_MEM <- mu_hat + test %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i
# RMSE of new predictions
model_1_rmse <- RMSE(predicted_ratings_MEM, test$rating)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie Effect Model",
                                     RMSE = model_1_rmse ))
```

With the movie effect added we have another RMSE in our results table and can see our RMSE has improved:

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615

User Effect

Next we look at the average rating given by each user. We take each user and calculate the average difference between their rating for films and the mean rating, after taking into account the movie effect, to arrive at a user effect factor which is saved in user_avgs as b_u.

```
# calculate a user factor,
# average difference between users scores and overall average
user_avgs <- train %>%
  left_join(movie_avgs, by='movieId') %>%
```

```
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))
# predictions based on user factor and movie factor
predicted_ratings_UEM <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  .$pred
# RMSE of model 2
model_2_rmse <- RMSE(predicted_ratings_UEM, test$rating)
# added to results table
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User Effects Model",
                                     RMSE = model_2_rmse ))
```

And with the user effect added to our model the RMSE has improved yet again:

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843

Genre Effect

Next we look at the effect of other ratings for films in the same combination of genres. We take each combination of genres and calculate the average difference between ratings for films in that genre and the expected rating given the overall average, the movie effect, and the user effect. We arrive at a genre effect factor which is saved in genre_avgs as b_g.

```
# calculate genre factor
# average difference between genre's score and overall average
genre_avgs <- train %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu_hat - b_i - b_u))
# predictions based on genre, user, and movie effect factors
predicted_ratings_GEM <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu_hat + b_i + b_u + b_g) %>%
  .$pred
# RMSE of model 3
model_3_rmse <- RMSE(predicted_ratings_GEM, test$rating)
# added to results table
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User + Genre Effects Model",
                                     RMSE = model_3_rmse ))
```

With the genre effect added we check against the test set to see if our RMSE has improved, and again we see a small improvement:

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843
Movie + User + Genre Effects Model	0.8643241

Time Since Release Effect

Next we look at the effect of time passing between a film's release and the rating being submitted. We group the ratings by the difference between release year and year of rating and calculate the average difference between the ratings and the expected rating given the overall average, movie effect, user effect, and genre effect. We arrive at a time since release effect factor which is saved in since_release_avgs as b_sr.

```
# calculate since release factor
since_release_avgs <- train %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  group_by(since_release) %>%
  summarize(b_sr = mean(rating - mu_hat - b_i - b_u - b_g))
# predictions based on since release, genre, user, and movie effect factors
predicted_ratings_SREM <- test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(since_release_avgs, by='since_release') %>%
  mutate(pred = mu_hat + b_i + b_u + b_g + b_sr) %>%
  .$pred
# RMSE of model 4
model_4_rmse <- RMSE(predicted_ratings_SREM, test$rating)
# added to results table
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User + Genre + Since Release Effects Model",
                                 RMSE = model_4_rmse ))
```

With the time since release effect added we again check the RMSE against the test set and see another small improvement:

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843
Movie + User + Genre Effects Model	0.8643241
Movie + User + Genre + Since Release Effects Model	0.8639307

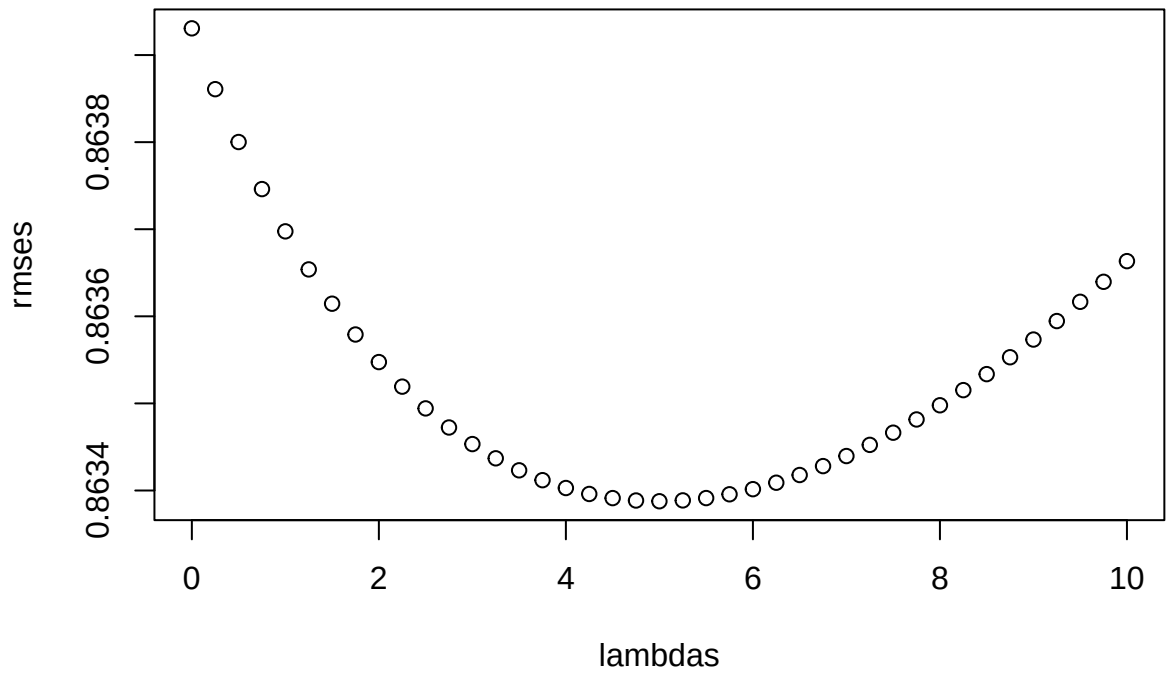
Regularization

In the final step of our model we regularize the data by applying a penalty factor to each piece of the model, lambda. We do not have to blindly pick λ; however, we can first tune for an ideal λ against the test set.

Note: this code will take several minutes to run.

```
# This code may take several minutes to run
# test lambdas for regularization against test set for best RMSE
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train$rating)
  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_g <- train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))
  b_sr <- train %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    group_by(since_release) %>%
    summarize(b_sr = sum(rating - b_i - b_u - b_g - mu)/(n()+l))
  predicted_ratings <-
    test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_sr, by = "since_release") %>%
    mutate(pred = mu + b_i + b_u + b_g + b_sr) %>%
    .$pred
  return(RMSE(predicted_ratings, test$rating))
})
```

We can examine the relationship between RMSE and λ using a chart:



We keep the best λ and the RMSE it produces is stored in the results table.

```
lambda <- lambdas[which.min(rmses)]
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularized Movie + All Effects Model",
                                 RMSE = min(rmses)))
```

method	RMSE
Just the average	1.0600537
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843
Movie + User + Genre Effects Model	0.8643241
Movie + User + Genre + Since Release Effects Model	0.8639307
Regularized Movie + All Effects Model	0.8633877

We can see from these results that regularization has again improved our RMSE. We can now go back to our old models and revise them to include our regularization factor (λ) before moving on to the final validation. Since we do not intend to further refine our model we will make this final adjustment using the full edx set.

```
# add the regularization factor into the models for movie, user, genre, and since release
# our tuning is complete so here we build the final model on the full edx set
edx_mu <- mean(edx$rating)
movie_avgs <- edx %>%
  group_by(movieId) %>%
```

```
  summarize(b_i = sum(rating - edx_mu)/(n()+lambda))
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - edx_mu - b_i)/(n()+lambda))
genre_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - edx_mu - b_i - b_u)/(n()+lambda))
since_release_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  group_by(since_release) %>%
  summarize(b_sr = sum(rating - edx_mu - b_i - b_u - b_g)/(n()+lambda))
```

Results and Final Validation

For our final validation we take all of our model inputs generated against the edx set and use them to attempt to predict the ratings of the validation set. We store these ratings as y_hat and then calculate the RMSE of these predictions saving the RMSE to our results table alongside a row with our goal RMSE for comparison.

```
# predict validation set with all 4 factors and lambda
y_hat <- validation %>%
  mutate(since_release = year(as_datetime(timestamp))-
           as.numeric(str_match(title,"\\((\\d{4})\\)$")[,2])) %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(since_release_avgs, by='since_release') %>%
  mutate(pred = mu_hat + b_i + b_u + b_g + b_sr) %>%
  .$pred
# calculate RMSE of prediction against validation set
RMSE(validation$rating,y_hat)
```

```
## [1] 0.8640157
```

```
# add target final results to table
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Target Final Result",
                                 RMSE = 0.8649000))
# add final results to table
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Final Model vs Validation RMSE",
                                    RMSE = RMSE(validation$rating,y_hat)))
```

method	RMSE
Just the average	1.0600537

method	RMSE
Movie Effect Model	0.9429615
Movie + User Effects Model	0.8646843
Movie + User + Genre Effects Model	0.8643241
Movie + User + Genre + Since Release Effects Model	0.8639307
Regularized Movie + All Effects Model	0.8633877
Target Final Result	0.8649000
Final Model vs Validation RMSE	0.8640157

With all of the factors in our model we are able to improve on our goal RMSE.

Conclusion

We examined a popular data set, explored some of the aspects of the underlying data to look for trends that we could exploit to build a model, and then constructed a model incorporating that analysis into our predictive model.

Our model began as a simple prediction of assigning the average score of all users and movies to each movie. We added a movie effect factor to capture that some films have better average ratings that others. We added a user effect factor to capture that some users are harder or easier on films than others. We added a genre effect factor to capture that some genres are perceived as more prestigious than others and get higher ratings. Finally we added a time since release effect factor to capture that the age of a film when it is watched affects the ratings a film tends to get. With all of these factors combined we were able to reduce our RMSE to 0.8640157 compared to our goal of 0.8649000.

Limitations

The model employed is an expansion on the linear model from the coursework. We were able to improve the RMSE over the linear model presented in Section 8 by introducing a couple of new factors, including one relatively novel one based on time since release, but the results are still limited by the techniques employed and could be improved on with a better model.

Several of the factors in this model rely on a user or film having a history of ratings to look back on. Predictions for new users or new films will be considerably less accurate as a result. In addition the model would need to be rerun frequently to keep the time sensitive factors in the model up to date such as each users rating history, the rating history of each film, or the age of a movie relative to the current date.

Future Work

There are many other widely adopted modeling systems that could be used to improve on these results such as Matrix Factorization or Principle Component Analysis. The winning model from the original Netflix challenge also uses Neighborhood Models which look at sets of similar users or similar items to help improve the accuracy of the model. The genre effects used in the analysis capture this to some degree but not to nearly as great an extent as was achieved in the winning result.

References

1. Rafael A. Irizarry (2022) Introduction to Data Science Data Analysis and Prediction Algorithms with R
2. Edwin Chen (2022) Winning the Netflix Prize: A summary