Movie Recommendation System

Michael Omelchenko

11/23/2020

Abstract

Recommendation systems are one of the cornerstone examples of machine learning algorithms. These recommendation systems have become common in everyday life. Some examples include Netflix predicting the rating a user would give a movie, Amazon predicting product preferences that a user would have, and more.

In this project, we will be examining arguably the most famous example of the recommendation system - The Netflix Competition. Netflix launched this competition in hopes of improving their recommendation system by 10%, with the winning prize being $1,000,000. In 2009, a winning team was awarded this prize, but the lasting impact is still felt in data science today.

Projects, such as this one, often emulate this challenge while teaching machine learning techniques to new data scientists. Because data from Netflix is not publicly available, the movielens dataset with 10,000,000 observations is used. Using some of the machine learning techniques learned throughout the class, such as Linear Regression and Regularization, we found a Root Mean Squared Error (RMSE) of 0.825.

Introduction

For the movie recommendation system, we will use the movielens dataset. The goal of this project is to create a recommendation system that predicts ratings for a given user and movie. We will use predictors such as movie, user, genre, and time in order to train our machine learning algorithms on a portion of the data (the training set). Then we will test our algorithms by predicting ratings on the other portion of the data (the test set). Various machine learning methods will be used including linear regression and regularization. For each method, we will calculate the Root Mean Squared Error (RMSE) to evaluate how accurate our predictions are. RMSE's near 1 are viewed as not good algorithms.

Methods/Analysis

This section explains the process and techniques used, including data creation, data cleaning, data visualization, and machine learning models.

Creation of Training and Test Sets

First we will start off by loading the libraries needed for this evaluation. In this project we will be using the tidyverse package for data manipulation, the caret package for partitioning the data into training and test sets,the lubridate package to manipulate dates within the data set, the knitr/kableExtra packages to visualize dataframes neatly, and the data.table library to read in the movielens data.

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

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

Next, we create the training and test sets for this machine learning algorithm. To do this we use the createPartition function in the caret package. Here we create a test index on the ratings column that is 10% of the dataset and then use this index to create the edx (training) and validation (test) sets. Please not if using R 3.6 or earlier there is a different line of code that has been commented out.

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

# Note: this process could take a couple of minutes
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

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

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

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


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


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

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

Data Exploration and Visualization

Focusing solely on the training set produced, we can explore the data to identify important predictors that may be useful for our machine learning algorithm.

First we examine the structure of the training set (denoted edx), and the test set (denoted validation). Because the training set and test set are made by partitioning the same data, they will have the same structure so only one is examined here.

```
str(edx)
```

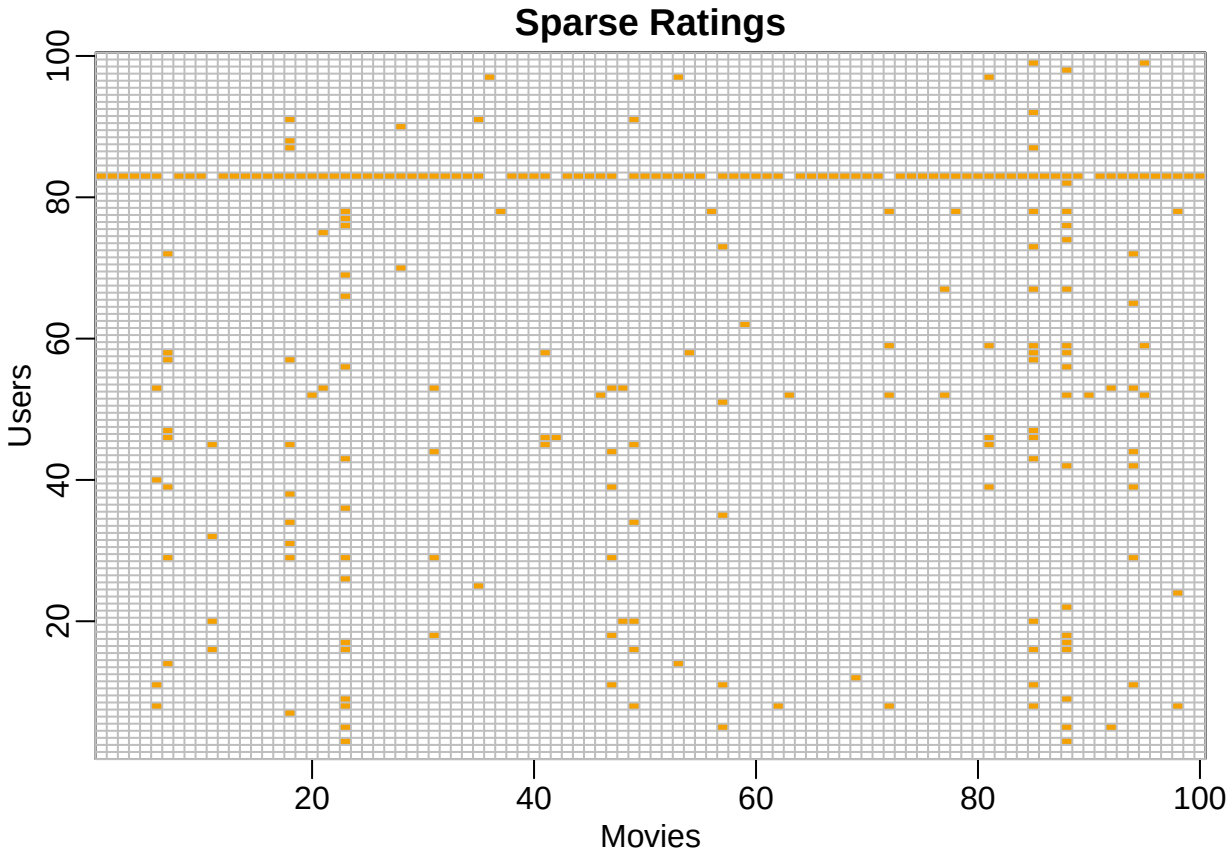
```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 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>
```

Ratings are Sparse

The first thing we notice is that movie ratings are fairly sparse. This makes sense, not every user has watched every movie and users in general do not rate every movie they watch.

To show this we sample 100 unique users and 100 random movies and plot the results. Movies that have not been rated are left as a blank grey square.

```
users <- sample(unique(edx$userId), 100)
rafalib::mypar()
edx %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>% select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users")
abline(h=0:100+0.5, v=0:100+0.5, col = "grey")
title("Sparse Ratings")
```



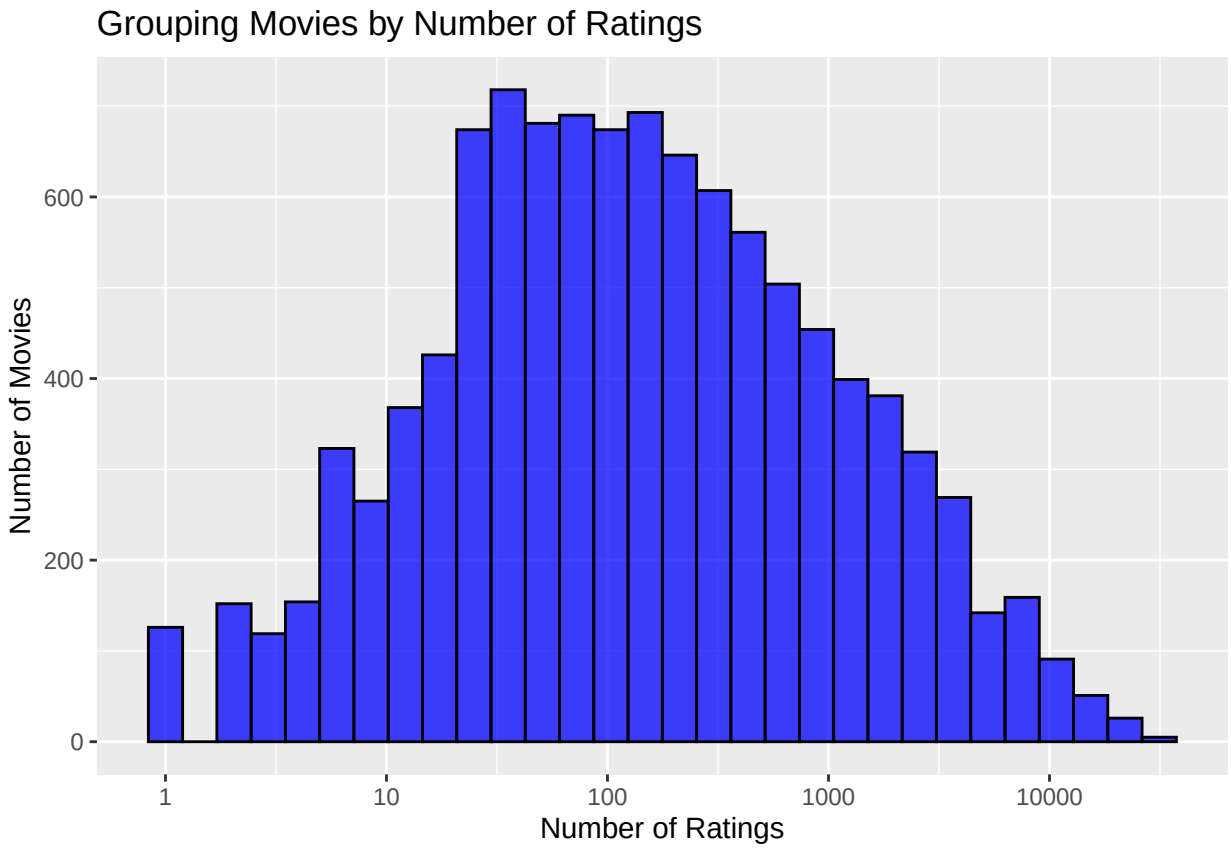
Movie Effect

The next thing we notice is that some movies get more ratings than others. Again, this is intuitive because some movies are big Hollywood successes while others are flops. We would expect more popular movies to have more views, and therefore more ratings.

```
edx %>%
  group_by(movieId) %>%
  summarize(n = n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black", fill = "blue", alpha = 0.75) +
  scale_x_log10() +
  ggtitle("Grouping Movies by Number of Ratings") +
  xlab("Number of Ratings") + ylab("Number of Movies")
```

Table 1: Movies with the Most Ratings

Title	Number of Ratings	Average Rating
Pulp Fiction (1994)	31362	4.154789
Forrest Gump (1994)	31079	4.012822
Silence of the Lambs, The (1991)	30382	4.204101
Jurassic Park (1993)	29360	3.663522
Shawshank Redemption, The (1994)	28015	4.455131



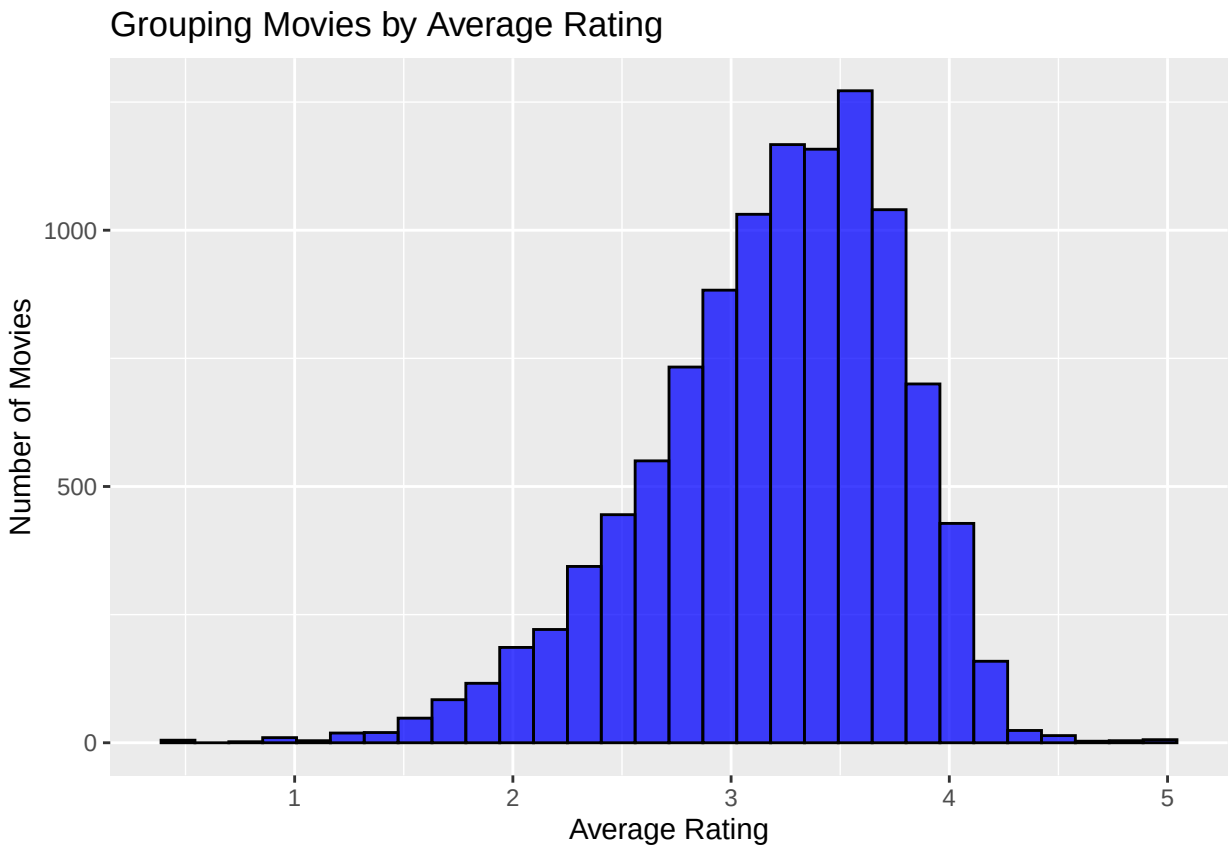
There are very few movies with over 10,000 ratings and very few movies with fewer than 10 ratings. Examining the top 5 movies with the most ratings we see these are infact very popular movies or big blockbuster hits.

```
tab <- edx %>%
  group_by(title) %>%
  summarize(Number_of_Ratings = n(), Average = mean(rating)) %>%
  arrange(desc(Number_of_Ratings)) %>%
  slice_head(n = 5)

colnames(tab) <- c("Title", "Number of Ratings", "Average Rating")


kable(tab, caption = "Movies with the Most Ratings", booktabs = T, linesep = "") %>%
  kable_styling(latex_options = "striped")
```

We also notice that certain movies have vastly different average ratings. Observing this wee see that there

clearly some movies are rated very poorly, some very favorably, and most in the middle.

```
edx %>% group_by(title) %>%
  summarize(Average = mean(rating)) %>%
  ggplot(aes(x = Average)) +
  geom_histogram(bins = 30, color = "black", fill = "blue", alpha = 0.75) +
  ggtitle("Grouping Movies by Average Rating") +
  xlab("Average Rating") + ylab("Number of Movies")
```

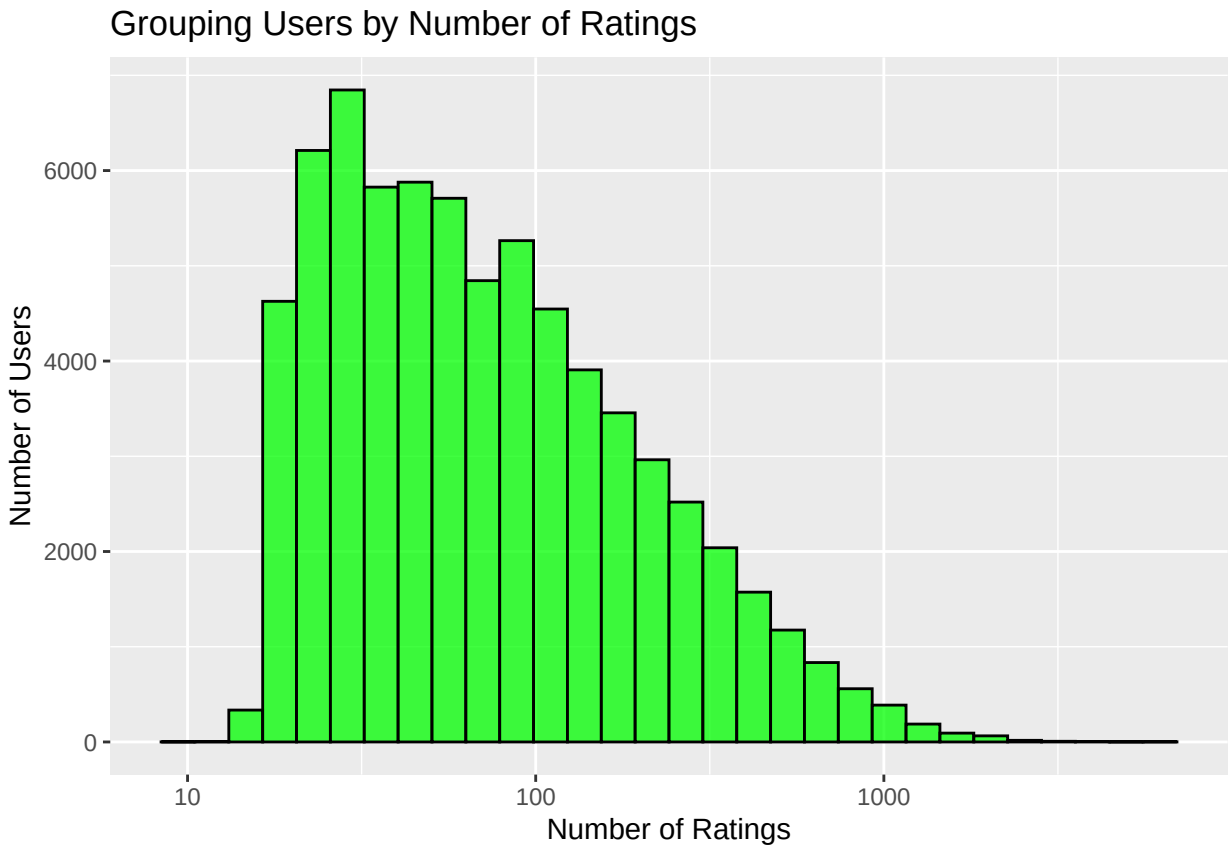


In summary, these effects show that the movie should be a relatively good predictor in our machine learning model because of the vast differences between movies and should be accounted for in our model.

User Effects

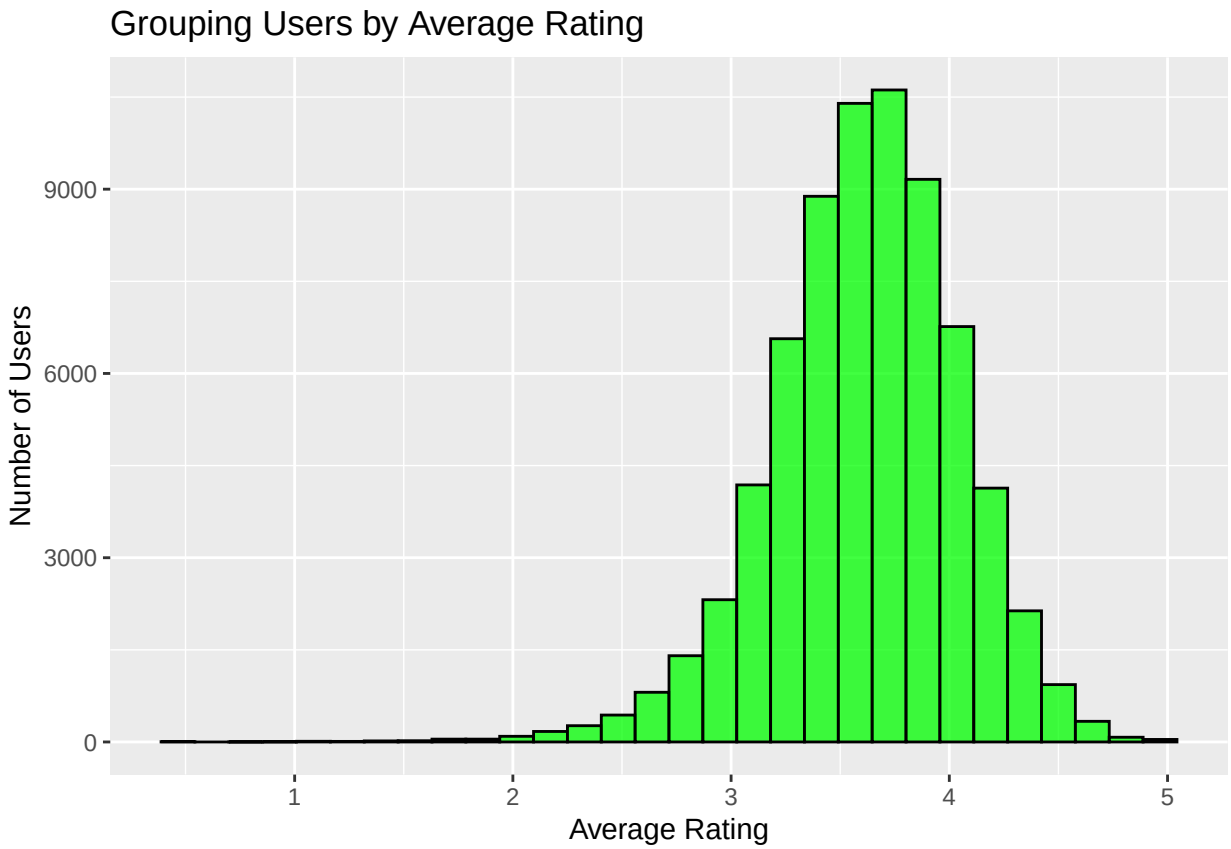
Now that we have observed that some some movies have more ratings than others, we will examine variances across users. Visualizing the data, we see that some users are more active in rating movies than others. Very few users have given less than 10 or more than 1000 ratings.

```
edx %>%
  group_by(userId) %>%
  summarize(n = n()) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black", fill = "green", alpha = 0.75) +
  scale_x_log10() +
  ggtitle("Grouping Users by Number of Ratings") +
  xlab("Number of Ratings") + ylab("Number of Users")
```



Similarly to the movies predictor, we can see that users rate movies very differently as well. Some users give every movie 5 stars (overly optimistic watchers) and other users give every movie 1 star (overly pesimistic users).

```
edx %>% group_by(userId) %>%
  summarize(Average = mean(rating)) %>%
  ggplot(aes(x = Average)) +
  geom_histogram(bins = 30, color = "black", fill = "green", alpha = 0.75) +
  ggtitle("Grouping Users by Average Rating") +
  xlab("Average Rating") + ylab("Number of Users")
```

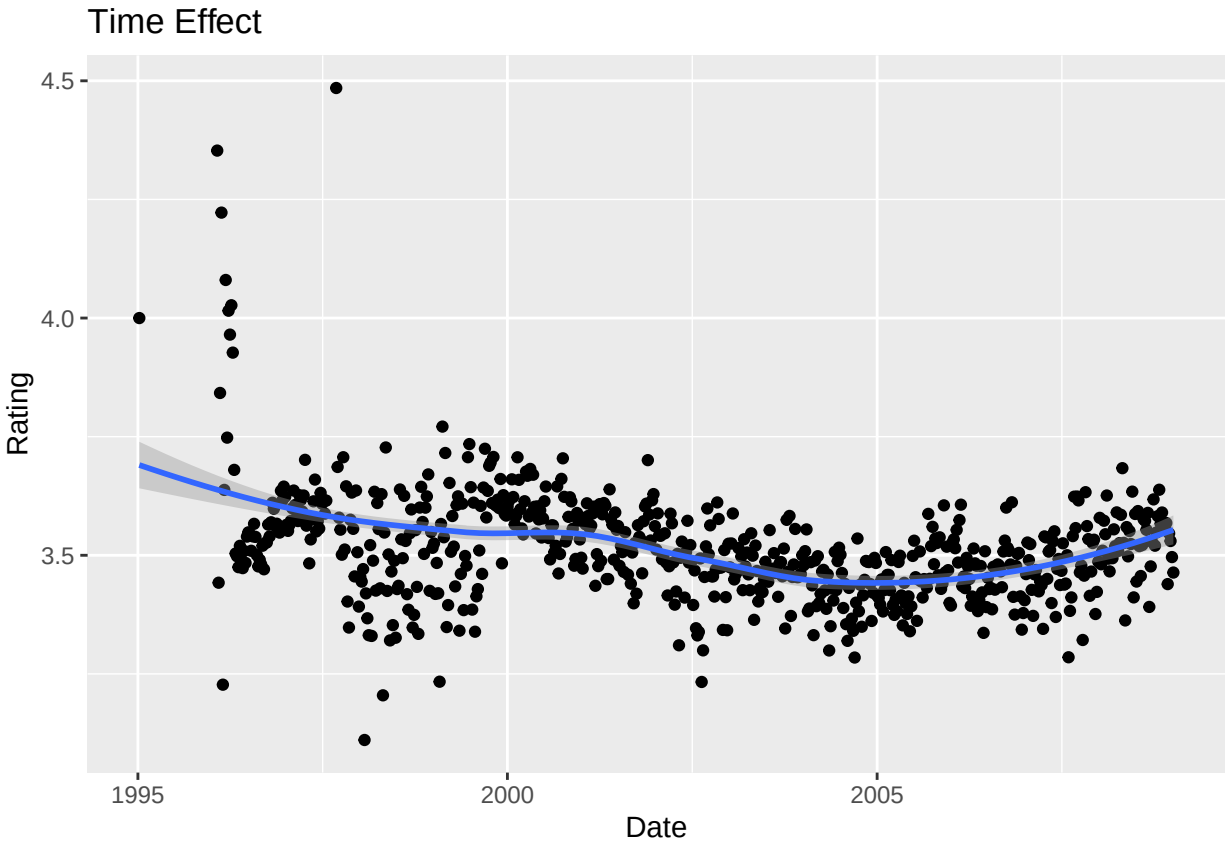


In summary, differences between users show that the user should be a relatively good predictor in our machine learning model.

Time Effect

We can also see that there is an observable effect of time on movie ratings. However, this effect appears to be relatively small (the line is almost horizontal, but slightly decreases as time goes on).

```
edx %>%
  mutate(date = as_datetime(timestamp)) %>%
  mutate(week = round_date(date, unit = "week")) %>%
  group_by(week) %>%
  summarize(rating = mean(rating)) %>%
  ggplot(aes(week, rating)) +
  geom_point() +
  geom_smooth() +
  ggtitle("Time Effect") +
  xlab("Date") + ylab("Rating")
```

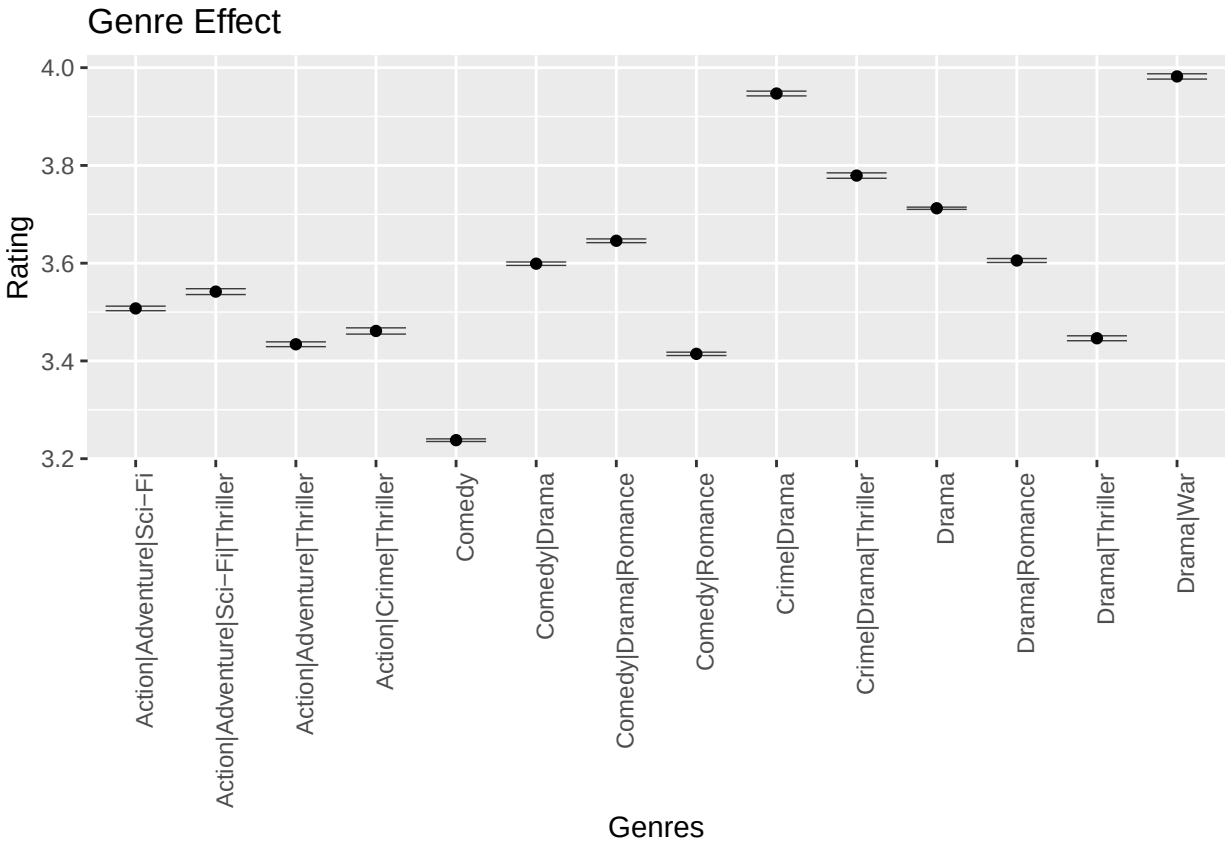


Time may be useful as a predictor but because of the almost horizontal nature of the loess line fitted to the data, we would not expect it to affect our model as much as the movie or user effect.

Genre Effect

Lastly, grouping by genre of movie, we can see that certain genres have higher ratings than others. Some movies fall under several genres. These genres will not be split for this visualization. Here we focus on genres with over 100,000 ratings for ease of reading.

```
edx %>% group_by(genres) %>%
  summarize(n = n(), avg = mean(rating), stdev = sd(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n > 100000) %>%
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar(width=0.75, colour="black", alpha=0.75, size=0.25) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ggtitle("Genre Effect") +
  xlab("Genres") + ylab("Rating")
```



We see that genre may be useful as a predictor, even when not split. This is because there appear to be differences between genres, and the standard deviation within each group appears to be small. However because most have a rating within 1 star of each other, we would not expect it to affect our model as much as the movie or user effect.

Evaluation Method

In order to evaluate how well our predictions are we need to determine how we will compare our predicted results to the actual results in the test set. In order to do that we will use the RMSE.

$$RMSE = \sqrt{\frac{\sum_{u,i}(y_{u,i} - yhat_{u,i})^2}{N}}$$

where:

- $y_{u,i}$ is the rating for movie i by user u
- $yhat_{u,i}$ is the rating for movie i by user u

Here we create a function that calculates the RMSE that we will use while to evaluate how well our model does at predicting ratings.

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

Results - Models Without Regularization

Below we will try different machine learning algorithms with the goal being to generate the model that has the lowest RMSE by only using the training set.

Naive Bayes

For our baseline case, we will make the most basic prediction. This prediction will be the a single value μ.

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

where:

- $Y_{u,i}$ is the rating for movie i by user u
- μ is the average overall rating
- ϵ is the independent errors sampled from the same distribution centered at zero

  By examining the forumla, we see that the overall average will give the lowest RMSE for guessing a single number. We see that using this approach we get an RMSE of about 1.06.

```
#getting the average overall
avg_rating_overall <- edx %>% summarize(mu = mean(rating)) %>% pull(mu)

#baseline of naive rmse i.e assuming average rating is prediction
naive_rmse <- RMSE(validation$rating, avg_rating_overall)
naive_rmse
```

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

Movie Bias Model

Based on our data visualization we saw that we should be able to do better using the movie as one of the predictors. We cannot do linear regression using "lm" because the dataset is so large that it would crash the computer trying to fit so many terms. Instead we will add another parameter b_i to our equation and calculate it.

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

where:

- b_i is the average rating for movie i

By rearranging the formula:

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

Using the new model we get an RMSE of 0.94 which is pretty large improvement.

```
#calculate the bias for each movie rating by subtracting the rating - mu
#because some movies are generally higher rated, and other movies are generally
#lower rated
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - avg_rating_overall))

pred_movies <- avg_rating_overall + validation %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

movies_rmse <- RMSE(validation$rating, pred_movies)
movies_rmse
```

```
## [1] 0.9439087
```

Movie + User Bias Model

During our data visualization, we also saw that the user had the next largest effect. Again because "lm" would crash the computer we attempt to estimate the effect by adding it to our model. Adding the parameter b_u to our equation and calculating the user effect.

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

where:

- b_u is the user effect

By rearranging the formula:

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

Using the model that takes into account both movie and user bias we get an RMSE of 0.865 which is another big improvement.

```
#we also need to calculate the bias for each user
#we do this by taking the rating and subtracting the average and movie bias
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - avg_rating_overall - b_i))

pred_user_movie <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = avg_rating_overall + b_i + b_u) %>%
  .$pred

movie_user_rmse <- RMSE(validation$rating, pred_user_movie)
movie_user_rmse
```

```
## [1] 0.8653488
```

Movies + Users + Genres Model

Next we add the genre effect into the model. From our data analysis we suspected this predictor would not have as large of an impact on our rating prediction but could help some. The new model becomes:

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

where:

- b_g is the genre effect

Like we stated above, we did not expect this predictor to have as big of an effect as movie or user, however it did slightly decrease the RMSE.

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

pred_movie_user_genre <- validation %>%
   left_join(movie_avgs, by='movieId') %>%
   left_join(user_avgs, by='userId') %>%
   left_join(genre_avgs, by='genres') %>%
```

```
   mutate(pred = avg_rating_overall + b_i + b_u + b_g) %>%
   .$pred

movie_user_genre_rmse <- RMSE(validation$rating, pred_movie_user_genre)
movie_user_genre_rmse
```

```
## [1] 0.8649469
```

Users + Movies + Genres + Time Model

Similarly, we add the time effect into our model.

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

where:

- b_w is the time effect

Here we need to do a little bit of data manipulation because the data is given as numeric POSIXt object with an origin date of "1970-01-01". We use the as_datetime function to get the date into a format we understand. Then we group the dates by week using the round_date function. Again we see a slightly improved RMSE, but by very little.

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

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

week_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  group_by(week) %>%
  summarize(b_w = mean(rating - avg_rating_overall - b_i - b_u - b_g))

pred_movie_user_genre_week <- validation %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(week_avgs, by='week') %>%
  mutate(pred = avg_rating_overall + b_i + b_u + b_g + b_w) %>%
  .$pred

movie_user_genre_week_rmse <- RMSE(validation$rating, pred_movie_user_genre_week)
movie_user_genre_week_rmse
```

```
## [1] 0.8648392
```

Results no Regularization

Putting all the results into a table we see that the best method thus far was the approach that took into account the movie, user, genre, and time effect.

```
result_no_reg <- data.frame("Model" = c("Naive Bayes","Movie Bias",
                                        "Movie + User Bias",
                                        "Movie + User + Genre Bias",
```

Table 2: Summarized Results

Model	RMSE
Naive Bayes	1.0612018
Movie Bias	0.9439087
Movie + User Bias	0.8653488
Movie + User + Genre Bias	0.8649469
Movie + User + Genre + Time Bias	0.8648392

```
                                        "Movie + User + Genre + Time Bias"),
                            "RMSE" = c(naive_rmse, movies_rmse,
                                       movie_user_rmse,
                                       movie_user_genre_rmse,
                                       movie_user_genre_week_rmse))
kable(result_no_reg, caption = "Summarized Results", booktabs = T, linesep = "") %>%
  kable_styling(latex_options = "striped")
```

Results - Regularized Model

In order to imrpove our results we want to use regularization. This method helps reduce variability of the effect by penalizing large effects that come from small samples. We can see these effects by examining the movies with the largest movie effect (b_i). It does not make sense for these movies to have a high b_i close to 1.5 because that would imply a perfect rating (~3.5 + 1.5) and these are obscure movies.

```
movie_titles <- edx %>%
     select(movieId, title) %>%
     distinct()

tab <- edx %>% dplyr::count(movieId) %>%
     left_join(movie_avgs) %>%
     left_join(movie_titles, by="movieId") %>%
     arrange(desc(b_i)) %>%
     select(title, b_i, n) %>%
     slice(1:10)
colnames(tab) <- c("Title", "bi", "Number of Ratings")

kable(tab, caption = "Movie Effects", booktabs = T, linesep = "") %>%
  kable_styling(latex_options = "striped")
```

In order to use regularization we will try to minimize the following equation:

$\frac{1}{N}\sum_{u,i}(y_{u,i} - \mu - b_i)^2 + \lambda \sum_i b_i^2$

You can see in the above equation the first term is the mean squared error and the second is the penalty term that gets larger when the b terms get larger.

Rearranging to solve for b_i:

$b_i(\lambda) = \frac{1}{\lambda + n_i}\sum_{u=1}^{n_i}(Y_{u,i} - \mu)$

where:

- n_i is the number of ratings for movie i
- λ is the tuning parameter we will have to choose. As lambda gets larger we shrink more.

We can add more terms to this equation and shrink them as well.

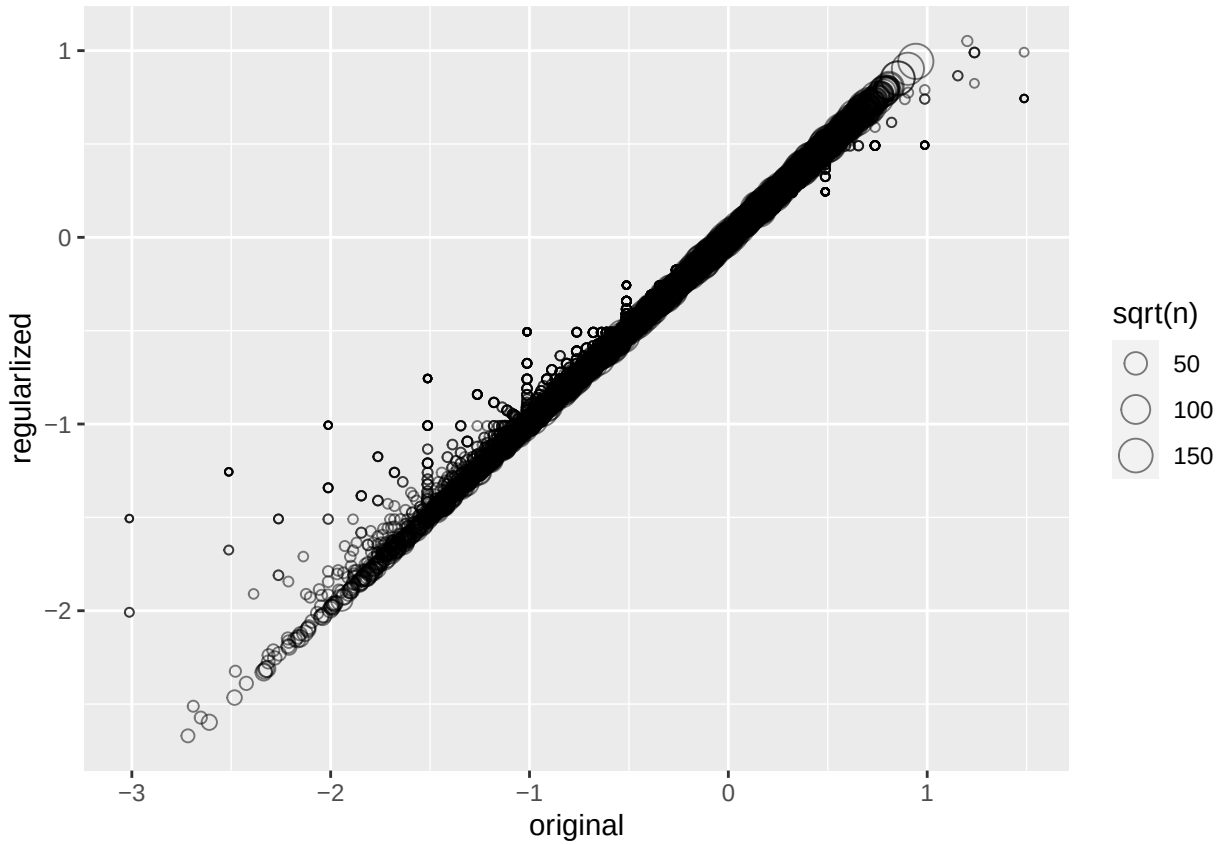
Table 3: Movie Effects

Title	bi	Number of Ratings
Hellhounds on My Trail (1999)	1.487535	1
Satan's Tango (Sátántangó) (1994)	1.487535	2
Shadows of Forgotten Ancestors (1964)	1.487535	1
Fighting Elegy (Kenka erejii) (1966)	1.487535	1
Sun Alley (Sonnenallee) (1999)	1.487535	1
Blue Light, The (Das Blaue Licht) (1932)	1.487535	1
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237535	4
Human Condition II, The (Ningen no joken II) (1959)	1.237535	4
Human Condition III, The (Ningen no joken III) (1961)	1.237535	4
Constantine's Sword (2007)	1.237535	2

$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i-b_u-b_g-b_w)^2+\lambda(\sum_i b_i^2+\sum_u b_u^2+\sum_g b_g^2+\sum_w b_w^2)$

To show the benefits of regularization we examine the example below in which we see what happens to the movie effect after regularization. Here we will pick a random lambda. We see from the plot as the circles get smaller (smaller n), the regularized values are shrunken towards zero.

```
lambda <- 1
movie_reg_avgs <- edx %>%
     group_by(movieId) %>%
     summarize(b_i = sum(rating - avg_rating_overall)/(n()+lambda), n_i = n())

data_frame(original = movie_avgs$b_i,
           regularlized = movie_reg_avgs$b_i,
           n = movie_reg_avgs$n_i) %>%
     ggplot(aes(original, regularlized, size=sqrt(n))) +
     geom_point(shape=1, alpha=0.5)
```




```
tab <- edx %>%
     dplyr::count(movieId) %>%
     left_join(movie_reg_avgs) %>%
     left_join(movie_titles, by="movieId") %>%
     arrange(desc(b_i)) %>%
     select(title, b_i, n) %>%
     slice(1:10)

colnames(tab) <- c("Title", "bi", "Number of Ratings")

kable(tab, caption = "Movie Effects with Regularization", booktabs = T, linesep = "") %>%
  kable_styling(latex_options = "striped")
```

Looking at the top movies after regularization we see that in general thse are much more well known movies with a higher number of ratings

Putting it all together

Putting it all together with regularized Movie, User, Genre, and Time biases into one model. Here we need to loop over various lambdas using sapply to pick the best one. After doing a very coarse first attempt (lambda changing by 1), we choose a lambda that is close to the minimum and do a finer analysis (lambda changing by 0.05). Since we cannot use the validation set to pick lambda we have to use different values on the training set to optimize the tuning parameter.

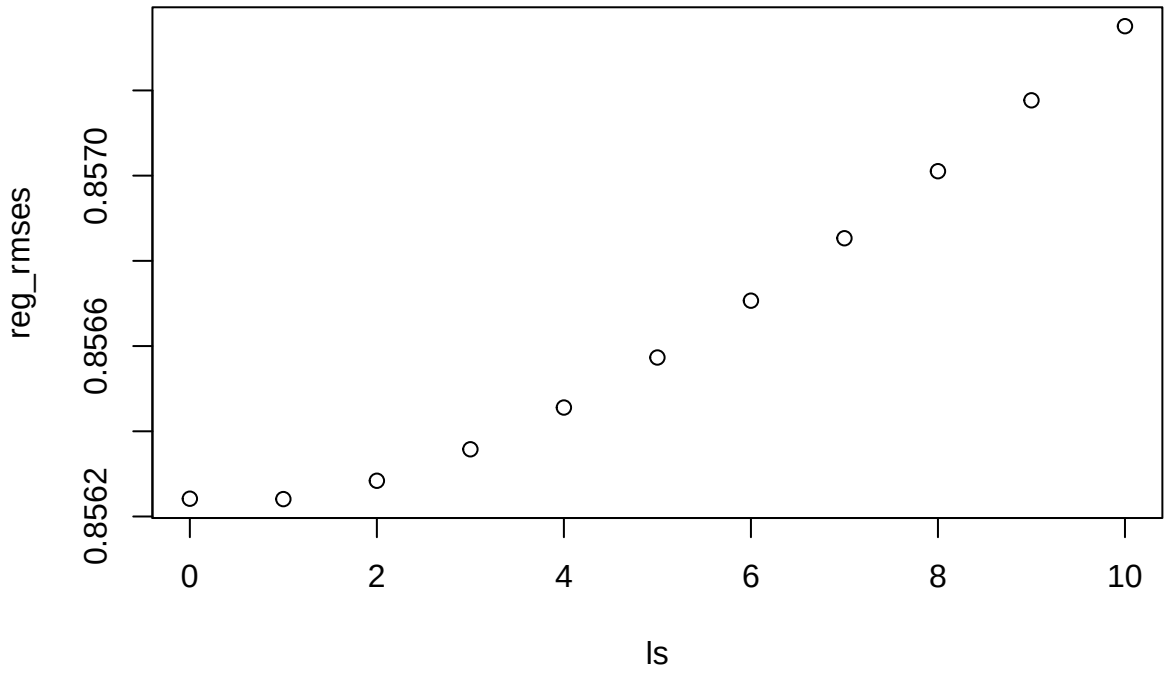
```
#run model with lambda spaced ever 1 to visualize where to focus in on
ls <- seq(0, 10, 1)

reg_rmses <- sapply(ls, function(l) {
```

Table 4: Movie Effects with Regularization

Title	bi	Number of Ratings
More (1998)	1.0515929	7
Satan's Tango (Sátántangó) (1994)	0.9916899	2
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	0.9900278	4
Human Condition II, The (Ningen no joken II) (1959)	0.9900278	4
Human Condition III, The (Ningen no joken III) (1961)	0.9900278	4
Shawshank Redemption, The (1994)	0.9426323	28015
Godfather, The (1972)	0.9028499	17747
I'm Starting From Three (Ricomincio da Tre) (1981)	0.8656511	3
Class, The (Entre les Murs) (2008)	0.8656511	3
Usual Suspects, The (1995)	0.8533490	21648

```
   # Calculate the regularized biases for movie, user, genre, and time

   movie_avgs_reg <- edx %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - avg_rating_overall) / (n() + l))

   user_avgs_reg <- edx %>%
      left_join(movie_avgs_reg, by='movieId') %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - avg_rating_overall - b_i) / (n() + l))

    genre_avgs_reg <- edx %>%
      left_join(movie_avgs_reg, by='movieId') %>%
      left_join(user_avgs_reg, by='userId') %>%
      group_by(genres) %>%
      summarize(b_g = sum(rating - avg_rating_overall - b_i - b_u) / (n() + l))

    week_avgs_reg <- edx %>%
      left_join(movie_avgs_reg, by='movieId') %>%
      left_join(user_avgs_reg, by='userId') %>%
      left_join(genre_avgs_reg, by='genres') %>%
      group_by(week) %>%
      summarize(b_w = sum(rating - avg_rating_overall - b_i - b_u - b_g) / (n() + l))


   pred_movie_user_genre_week_reg <- edx %>%
      left_join(movie_avgs_reg, by='movieId') %>%
      left_join(user_avgs_reg, by='userId') %>%
      left_join(genre_avgs_reg, by='genres') %>%
      left_join(week_avgs_reg, by='week') %>%
      mutate(pred = avg_rating_overall + b_i + b_u + b_g + b_w) %>%
      .$pred

   RMSE(edx$rating, pred_movie_user_genre_week_reg)
})

# pick the lambda to focus region around
```

```r
plot(ls, reg_rmses)
```

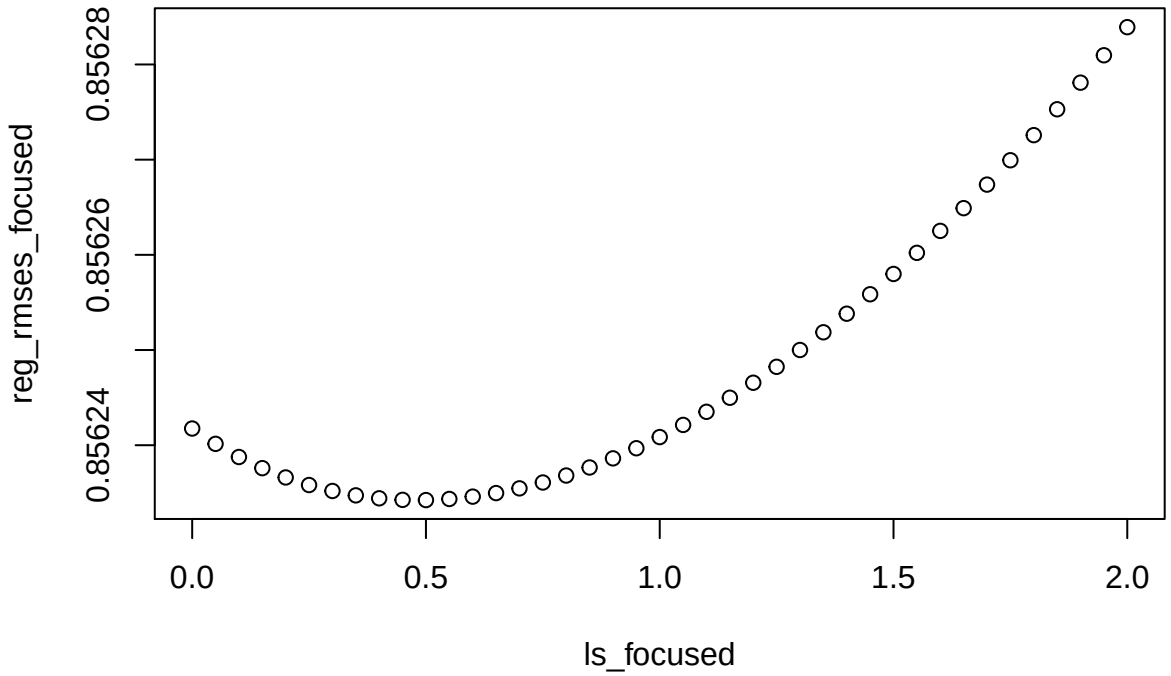



```r
l_first <- ls[which.min(reg_rmses)]
```

Using the coarse approach we see from the plot the minimum lambda is around 1. Below we test a new set of lambdas surrounding this point to use for our final model.

```r
#focusing region based on coarse lambda.
ls_focused <- seq(l_first - 1, l_first + 1, 0.05)

reg_rmses_focused <- sapply(ls_focused, function(l) {

   # Calculate the regularized biases for movie, user, genre, and time

   movie_avgs_reg <- edx %>%
      group_by(movieId) %>%
      summarize(b_i = sum(rating - avg_rating_overall) / (n() + l))

   user_avgs_reg <- edx %>%
      left_join(movie_avgs_reg, by='movieId') %>%
      group_by(userId) %>%
      summarize(b_u = sum(rating - avg_rating_overall - b_i) / (n() + l))

    genre_avgs_reg <- edx %>%
      left_join(movie_avgs_reg, by='movieId') %>%
      left_join(user_avgs_reg, by='userId') %>%
      group_by(genres) %>%
      summarize(b_g = sum(rating - avg_rating_overall - b_i - b_u) / (n() + l))

    week_avgs_reg <- edx %>%
      left_join(movie_avgs_reg, by='movieId') %>%
      left_join(user_avgs_reg, by='userId') %>%
      left_join(genre_avgs_reg, by='genres') %>%
```

```
      group_by(week) %>%
      summarize(b_w = sum(rating - avg_rating_overall - b_i - b_u - b_g) / (n() + l))

   #predict the movie ratings on edx set
   pred_movie_user_genre_week_reg <- edx %>%
      left_join(movie_avgs_reg, by='movieId') %>%
      left_join(user_avgs_reg, by='userId') %>%
      left_join(genre_avgs_reg, by='genres') %>%
      left_join(week_avgs_reg, by='week') %>%
      mutate(pred = avg_rating_overall + b_i + b_u + b_g + b_w) %>%
      .$pred

   #calculate RMSE
   RMSE(edx$rating, pred_movie_user_genre_week_reg)
})

# pick the final lambda
plot(ls_focused, reg_rmses_focused)
```




```
l_final <- ls_focused[which.min(reg_rmses_focused)]
l_final
```

```
## [1] 0.5
```

Now we need to rerun the regularization model with the optimum lambda (0.5) on the validation set to get our predictions.

```
# rerun the regularization on the validation set with final lamda to generate the RMSE
reg_rmse_validation <- sapply(l_final, function(l) {

   # Calculate the regularized biases for movie, user, genre, and time
   movie_avgs_reg <- validation %>%
      group_by(movieId) %>%
```

Table 5: Summarized Results

Model	RMSE
Naive Bayes	1.0612018
Movie Bias	0.9439087
Movie + User Bias	0.8653488
Movie + User + Genre Bias	0.8649469
Movie + User + Genre + Time Bias	0.8648392
Regularized Movie + User + Genre + Time Bias	0.8252400

```
    summarize(b_i = sum(rating - avg_rating_overall) / (n() + l))

  user_avgs_reg <- validation %>%
    left_join(movie_avgs_reg, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - avg_rating_overall - b_i) / (n() + l))

   genre_avgs_reg <- validation %>%
    left_join(movie_avgs_reg, by='movieId') %>%
    left_join(user_avgs_reg, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - avg_rating_overall - b_i - b_u) / (n() + l))

   week_avgs_reg <- validation %>%
    left_join(movie_avgs_reg, by='movieId') %>%
    left_join(user_avgs_reg, by='userId') %>%
    left_join(genre_avgs_reg, by='genres') %>%
    group_by(week) %>%
    summarize(b_w = sum(rating - avg_rating_overall - b_i - b_u - b_g) / (n() + l))

  #predict the movie ratings on validation set
  pred_movie_user_genre_week_reg <- validation %>%
    left_join(movie_avgs_reg, by='movieId') %>%
    left_join(user_avgs_reg, by='userId') %>%
    left_join(genre_avgs_reg, by='genres') %>%
    left_join(week_avgs_reg, by='week') %>%
    mutate(pred = avg_rating_overall + b_i + b_u + b_g + b_w) %>%
    .$pred

  #calculate RMSE
  RMSE(validation$rating, pred_movie_user_genre_week_reg)
})
```

Results Regularization

Putting all the results into a table we see that the best method was the approach that took into account the movie, user, genre, and time effects and regularized each effect. An RMSE of 0.825 is a significant improvement from the result without regularization.

```
results <- result_no_reg %>% add_row("Model" = "Regularized Movie + User + Genre + Time Bias", "RMSE" =
kable(results, caption = "Summarized Results", booktabs = T, linesep = "") %>%
  kable_styling(latex_options = "striped")
```

Conclusion

This project has examined several machine learning algorithms to predict movie ratings for the movielens data. Through data visualization, we saw that the movie and user effect would likely be the best predictors. However, there also appeared to be a small effect based on genre and time. Using the training set, several linear regression algorithms were trained. Evaluating these models performance based on RMSE showed our intuitions from visualization were correct. We saw the biggest improvement in model performance from the movie and user bias, but still achieved small improvements after including time and genre.
Further data analysis showed that without regularization, large effects were seen from small sample sizes. After penalizing those effects with a smaller sample size we saw a large improvement in model performance. Using the final model, we were able to achieve and RMSE of 0.825, a marked improvement over the linear regression models. Future work includes using Matrix Factorization and Principle Component Analysis in order to further decrease the RMSE. Additional machine learning techniques such as that are commonly used in recommendation algorithms should also be explored.

References

Formulas for linear regression, regularization, RMSE, portions of the data data visualization were adapted from Rafael Irizarry's "Introduction to Data Science" book. Many thanks to Professor Irizarry, classmates, and staff for help throughout this R Data Science Professional Certificate Course.