Movie Recommendation Capstone Project

Kevin Nolasco

9/15/2020

Introduction

The objective of this project is to predict movie ratings for various users using the **MovieLens Dataset**. The accuracy of the model will be based on the RMSE (root mean squared estimate). The dataset has 9,000,055 rows and 6 columns. The columns are:

- userId: The class of userId is *integer*.
- movieId: The class of movieId is *numeric*.
- rating: The class of rating is *numeric*.
- timestamp: The class of timestamp is *integer*.
- title: The class of title is *character*.
- genres: The class of genres is *character*.

There are 10,677 different movies and 69,878 different users in the dataset.

Methods/Analysis

First, we load MovieLens dataset and create the edx and validation sets.

```r
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")

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

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

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

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

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



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

```r
# 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)` instead
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)
```

The first thing to do is to split the edx dataset into test and training sets using the caret package. Also, we use a semi-join to ensure that the movies and users in the training set are also in the test set.

```r
library(caret)
test_index <- createDataPartition(edx$rating, times = 1, p = 0.2, list = FALSE)
test_set <- edx[test_index,]
train_set <- edx[-test_index,]

test_set <- test_set %>% semi_join(train_set, by = "movieId")
test_set <- test_set %>% semi_join(train_set, by = "userId")
```

Next, I wanted to see if there were any NA's left in the ratings of training set. This is important because NA's can negatively affect our analysis later.

```r
sum(is.na(train_set$rating))
```

```
## [1] 0
```

Since the model will be judged based on the RMSE, we create a function that caluculates the RMSE.

```r
RMSE <- function(actual, predicted){
  sqrt(mean((actual - predicted)^2))
}
```

Initial Models

Movie Effects

The first model that will be tested is a model that is based on the movie effects. Some movies are rated differently that others and this model will take that into account.

```r
mu <- mean(train_set$rating)

movie_effects <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

#now we make our prediction using mean of movie ratings and movie effects
```

```
pred_1 <- mu + test_set %>%
  left_join(movie_effects, by = "movieId") %>%
  pull(b_i)

rmse1 <- RMSE(test_set$rating, pred_1)
rmse1
```

```
## [1] 0.9437144
```

With this simple model, we obtain a RMSE of 0.944.

Movie + User Effects

To improve this model and try to obtain a smaller RMSE, we consider a model that includes user effects. Some users rate movies differently than others because of their own preferences.

```
user_effects <- train_set %>%
  left_join(movie_effects, by = "movieId") %>% #we need to join the tables in order to have access to b
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

pred_2 <- mu + test_set %>%
  left_join(movie_effects, by = "movieId") %>%
  left_join(user_effects, by = "userId") %>%
  mutate(bu_bi = b_u + b_i) %>%
  pull(bu_bi)

rmse2 <- RMSE(test_set$rating, pred_2)
rmse2
```

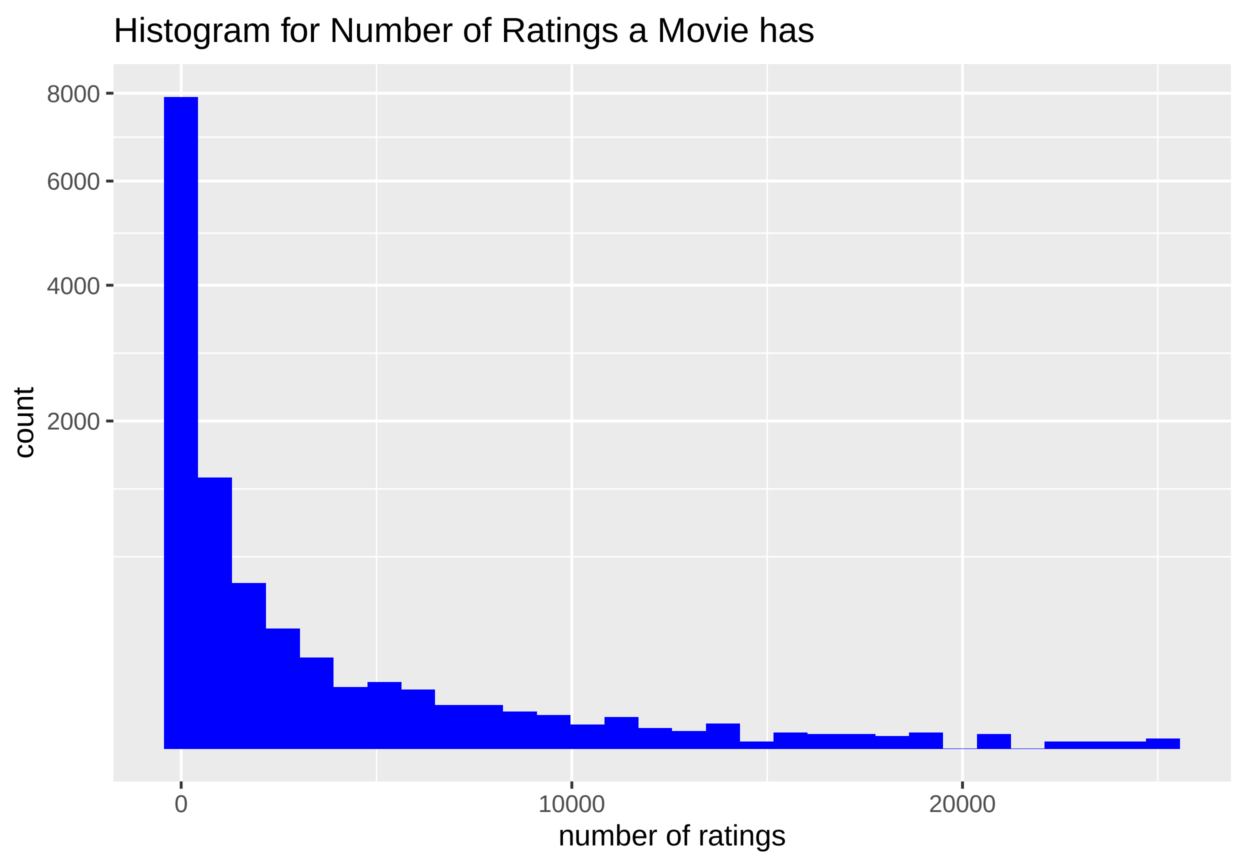
```
## [1] 0.8661625
```

By including the user effects, the RMSE reduces to 0.866. This is definitely better than the last model but we would still prefer a more accurate model.

Regularization Models

Regularized Movie Effects

Let's look at the number of times that movies are rated. (NOTE: The y-axis was scaled by a square root.)



It is clear that a many movies were rated a few times. Also, few movies were rated many times. To account for this, we will use regularization by movie effects.

The idea behind regularization is to minimize the least squares estimate plus a penalty term. The value that minimizes this estimate is λ. The value of λ is not yet known, so we use cross validation to find the λ that minimizes this estimate. It is appropriate to use regularization because some movies were given great ratings very few times.

```
lambdas <- seq(0,10,0.1)

sums <- train_set %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), nums = n())

rmse_cv <- sapply(lambdas, function(lam){#this function makes predictions using different values of lam
  predictions <- test_set %>%
    left_join(sums, by = "movieId") %>%
    mutate(b_i = s/(nums + lam)) %>%
    mutate(preds = mu + b_i) %>%
    pull(preds)
  return(RMSE(test_set$rating, predictions))
})


lambdas[which.min(rmse_cv)]
```

```
## [1] 1.7
```

Through this cross-validation, we see that $\lambda = 1.7$ is optimal for this model. Next we apply regularization of movie effects using $\lambda = 1.7$.

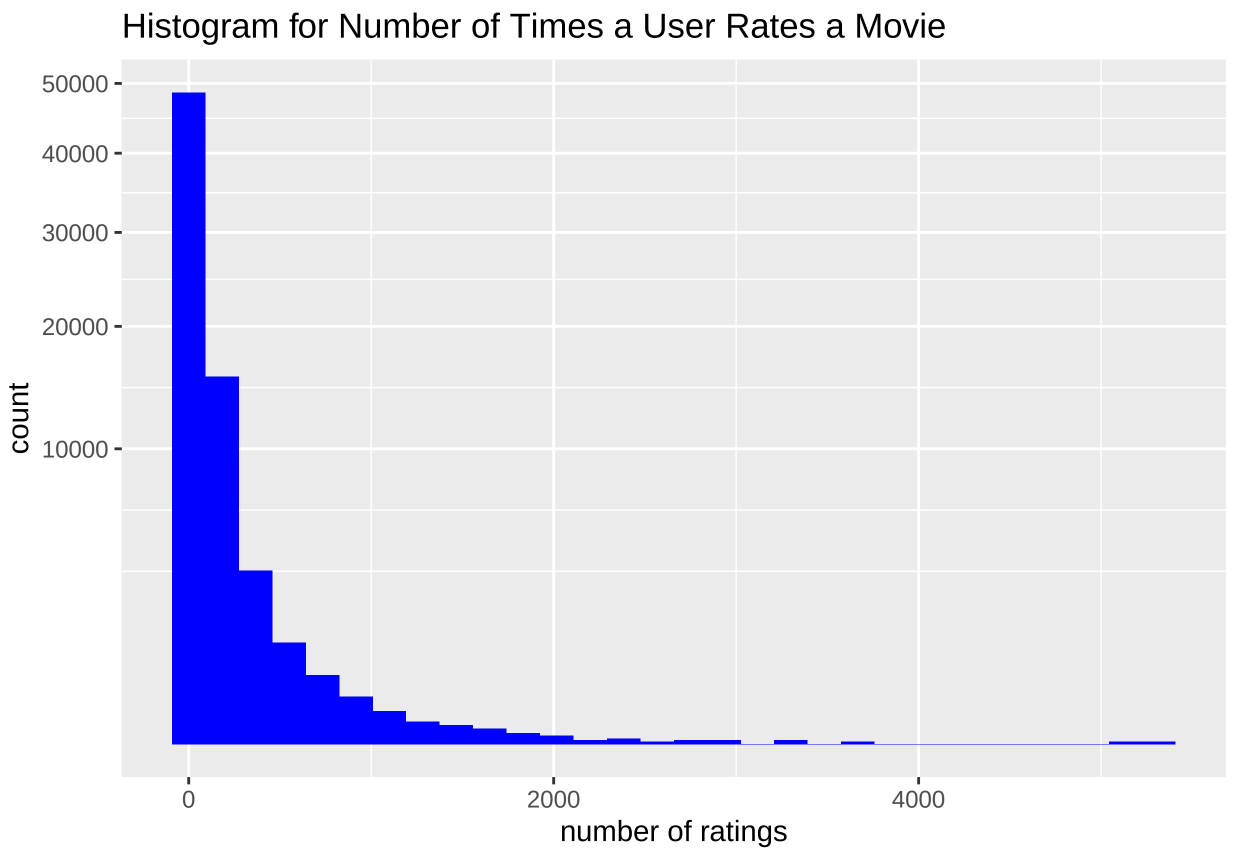
```
lambda <- 1.7

reg_movie_effects <- train_set %>%
  group_by(movieId) %>%
  summarize(reg_bi = sum(rating - mu)/(n() + lambda))

pred_3 <- test_set %>%
  left_join(reg_movie_effects, by = "movieId") %>%
  mutate(preds = mu + reg_bi) %>%
  pull(preds)

rmse3 <- RMSE(test_set$rating, pred_3)
rmse3
```

```
## [1] 0.9436775
```

This model gives an RMSE of 0.944. This is not as effective as I hoped for.

Regularized Movie + User Effects

Now, let's look at the number of times that users rate movies. (NOTE: The y-axis was scaled by a square root.)



It is appropriate to use regularization for user effects because some users did not rate movies as often as others. First, we must find the optimal λ for the regularized movie and user effects.

```r
rmses_cv2 <- sapply(lambdas, function(lam){
  reg_movie <- train_set %>%
    group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(lam + n())) #reg_movie will contain bi

  reg_user <- train_set %>%
    left_join(reg_movie, by = "movieId") %>% #need to join so that we have access to bi
    group_by(userId) %>%
    summarize(bu = sum(rating - mu - bi)/(lam + n()))

  predictions <- test_set %>%
    left_join(reg_movie, by = "movieId") %>% #need to join so we have access to both bi and bu
    left_join(reg_user, by = "userId") %>%
    mutate(preds = mu + bi + bu) %>%
    pull(preds)

  return(RMSE(test_set$rating, predictions))
})

lambdas[which.min(rmses_cv2)]
```

```
## [1] 4.7
```

Through this cross-validation, we see that $\lambda = 4.7$ is optimal for this model. Next we apply regularization of movie and user effects using $\lambda = 4.7$.

```r
lambda2 = 4.7

reg_movie <- train_set %>%
  group_by(movieId) %>%
  summarize(bi = sum(rating - mu)/(lambda2 + n())) #reg_movie will contain bi

reg_user <- train_set %>%
  left_join(reg_movie, by = "movieId") %>% #need to join so that we have access to bi
  group_by(userId) %>%
  summarize(bu = sum(rating - mu - bi)/(lambda2 + n()))

pred_4 <- test_set %>%
  left_join(reg_movie, by = "movieId") %>% #need to join so we have access to both bi and bu
  left_join(reg_user, by = "userId") %>%
  mutate(preds = mu + bi + bu) %>%
  pull(preds)

rmse4 <- RMSE(test_set$rating, pred_4)
rmse4
```

```
## [1] 0.8655424
```

This model gives an RMSE of 0.8655424. This is the lowest RMSE! But I would like to do better.

Ensemble? Next, I create an ensemble of the four methods used so far: movie effects, movie + user effects, regularized movie effects, and regularized movie + user effects.

```r
pred_5 <- (pred_1 + pred_2 + pred_3 + pred_4)/4

rmse5 <- RMSE(test_set$rating, pred_5)
```

```
rmse5
```

```
## [1] 0.8847612
```

This ensemble gave an RMSE of 0.8847. This is more than the previous model, so I don't think the ensemble is working.

Regularized Movie + User + Genre Effects

So far the regularized movie + user effects gave the best results. Perhaps including the regularized genre effect will help the RMSE. Some genres are watched and favored more than others so there must be some influence based on genre. As before, we start by using cross-validation to find the optimal λ for this model.

```
rmses_cv3 <- sapply(lambdas, function(lam){
  reg_movie <- train_set %>%
    group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(lam + n())) #reg_movie will contain bi

  reg_user <- train_set %>%
    left_join(reg_movie, by = "movieId") %>% #need to join so that we have access to bi
    group_by(userId) %>%
    summarize(bu = sum(rating - mu - bi)/(lam + n()))

  reg_genre <- train_set %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - bi - bu)/(lam + n()))

  predictions <- test_set %>%
    left_join(reg_movie, by = "movieId") %>% #need to join so we have access to both bi and bu
    left_join(reg_user, by = "userId") %>%
    left_join(reg_genre, by = "genres") %>%
    mutate(preds = mu + bi + bu + b_g) %>%
    pull(preds)

  return(RMSE(test_set$rating, predictions))
})



lambdas[which.min(rmses_cv3)]
```

```
## [1] 4.5
```

Through this cross-validation, we see that $\lambda = 4.5$ is optimal for this model. Next we apply regularization of movie, user, and genre effects using $\lambda = 4.5$.

```
lambda3 <- 4.5
#now we run the model with the optimal lambda

reg_movie <- train_set %>%
  group_by(movieId) %>%
  summarize(bi = sum(rating - mu)/(lambda3 + n())) #reg_movie will contain bi

reg_user <- train_set %>%
  left_join(reg_movie, by = "movieId") %>% #need to join so that we have access to bi
  group_by(userId) %>%
  summarize(bu = sum(rating - mu - bi)/(lambda3 + n()))

reg_genre <- train_set %>%
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - bi - bu)/(lambda3 + n()))

pred_6 <- test_set %>%
  left_join(reg_movie, by = "movieId") %>% #need to join so we have access to both bi and bu
  left_join(reg_user, by = "userId") %>%
  left_join(reg_genre, by = "genres") %>%
  mutate(preds = mu + bi + bu + b_g) %>%
  pull(preds)

rmse6 <- RMSE(test_set$rating, pred_6)
rmse6
```

```
## [1] 0.8652434
```

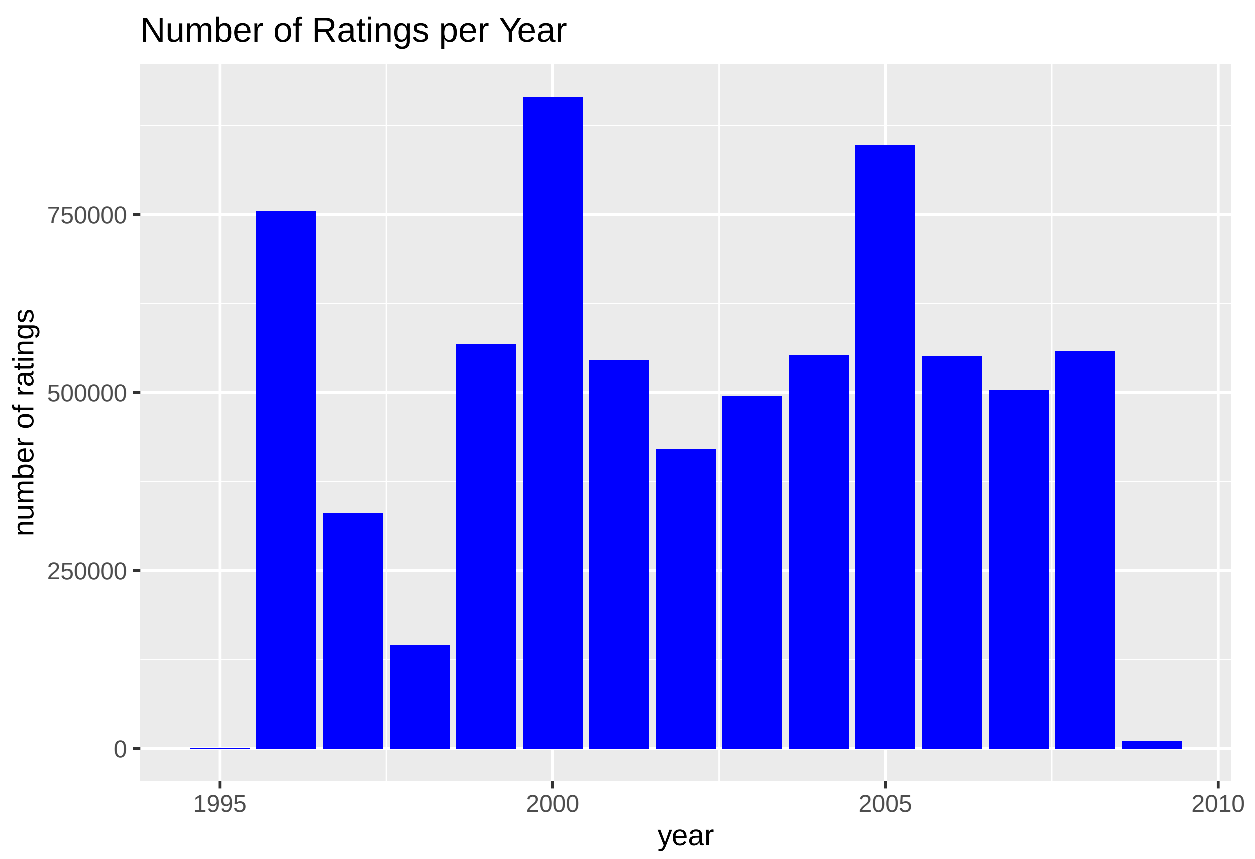
This model reproduced a RMSE of 0.8652434.

Regularized Movie + User + Genre + Year

Although this is better than regularized movie + user effects, I am curious if regularizing by month and year would make a difference. First, I create a month and year column for the test and training set using the Lubridate package.

```
library(lubridate)
train_set <- train_set %>%
  mutate(date = as_datetime(timestamp), year = year(date), month = month(date))

test_set <- test_set %>%
  mutate(date = as_datetime(timestamp), year = year(date), month = month(date))
```

First, let’s explore the variability in the number of ratings per year.



As you can see from the figure above, there isn't that much variability in the number of ratings per year, but it may contribute to the model.

Next, I used cross-validation to find the optimal λ for the regularized movie + user + genre + year effects.

```r
lambdas <- seq(0,10,0.25)
rmses_cv4 <- sapply(lambdas, function(lam){
  reg_movie <- train_set %>%
    group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(lam + n())) #reg_movie will contain bi

  reg_user <- train_set %>%
    left_join(reg_movie, by = "movieId") %>% #need to join so that we have access to bi
    group_by(userId) %>%
    summarize(bu = sum(rating - mu - bi)/(lam + n()))

  reg_genre <- train_set %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - bi - bu)/(lam + n()))

  reg_year <- train_set %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
    left_join(reg_genre, by = "genres") %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - mu - bi - bu - b_g)/(lam + n()))
  predictions <- test_set %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
    left_join(reg_genre, by = "genres") %>%
    left_join(reg_year, by = "year") %>%
    mutate(preds = mu + bi + bu + b_g + b_y) %>%
    pull(preds)

  return(RMSE(test_set$rating, predictions))
})

lambda <- lambdas[which.min(rmses_cv4)]
```

Through this cross-validation, we see that $\lambda = 4.5$ is optimal for this model. Next we apply regularization of movie, user, genre, and year effects using $\lambda = 4.5$.

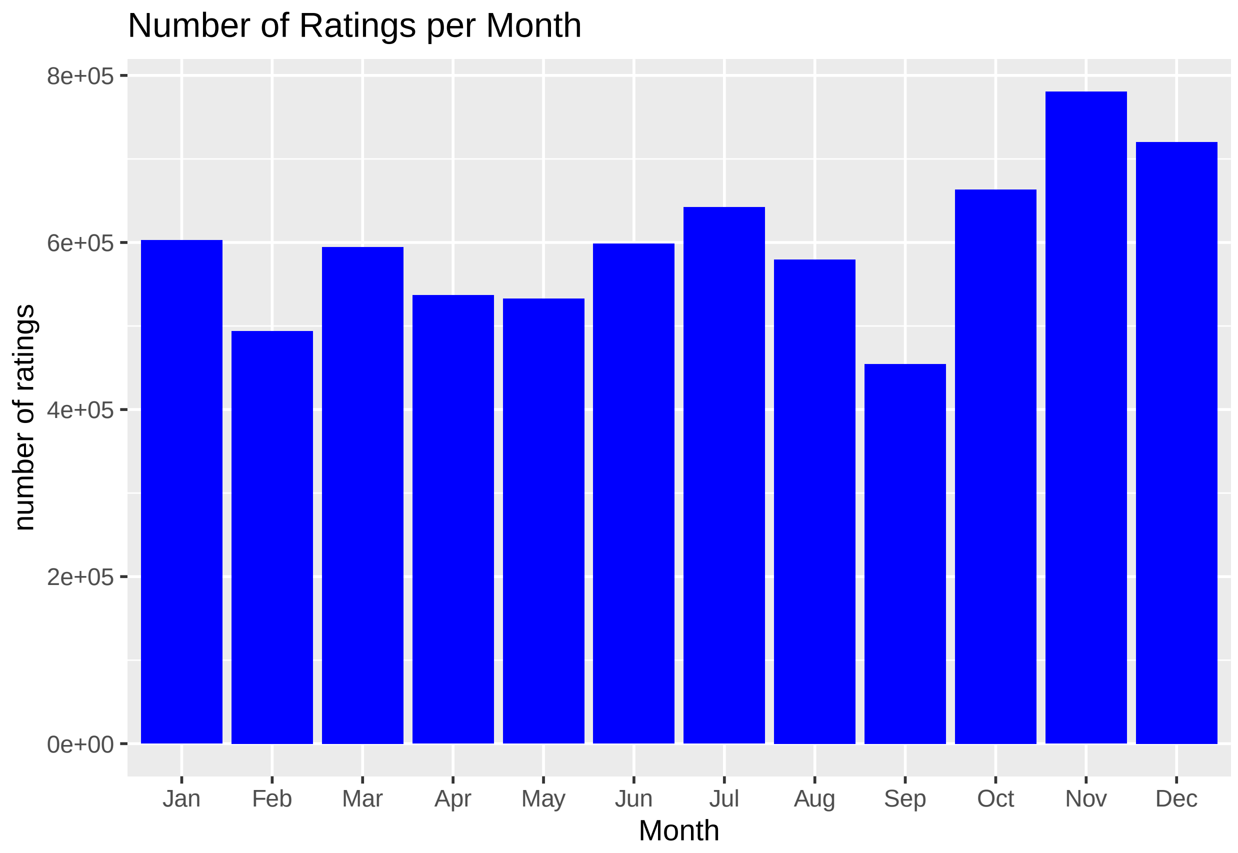
```
reg_movie <- train_set %>%
  group_by(movieId) %>%
  summarize(bi = sum(rating - mu)/(lambda + n())) #reg_movie will contain bi

reg_user <- train_set %>%
  left_join(reg_movie, by = "movieId") %>% #need to join so that we have access to bi
  group_by(userId) %>%
  summarize(bu = sum(rating - mu - bi)/(lambda + n()))

reg_genre <- train_set %>%
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - bi - bu)/(lambda + n()))

reg_year <- train_set %>%
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  left_join(reg_genre, by = "genres") %>%
  group_by(year) %>%
  summarize(b_y = sum(rating - mu - bi - bu - b_g)/(lambda + n()))

pred7 <- test_set %>%
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  left_join(reg_genre, by = "genres") %>%
  left_join(reg_year, by = "year") %>%
  mutate(preds = mu + bi + bu + b_g + b_y) %>%
  pull(preds)

rmse7 <- RMSE(test_set$rating, pred7)
rmse7
```

```
## [1] 0.8651889
```

This model achieved a RMSE of 0.8651899. This is pretty low! But I am curious whether the month will make a difference.

Regularized Movie + User + Genre + Year + Month Effects

Let's explore the variability in the number of ratings per month.



Again, there isn't that much variability in the number of ratings per month but it may still make a difference in the model.

So I run cross-validation again to find the optimal λ for the regularized movie + user + genre + year + month model.

```
rmses_cv5 <- sapply(lambdas, function(lam){
  reg_movie <- train_set %>%
    group_by(movieId) %>%
    summarize(bi = sum(rating - mu)/(lam + n())) #reg_movie will contain bi

  reg_user <- train_set %>%
    left_join(reg_movie, by = "movieId") %>% #need to join so that we have access to bi
    group_by(userId) %>%
    summarize(bu = sum(rating - mu - bi)/(lam + n()))

  reg_genre <- train_set %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - bi - bu)/(lam + n()))

  reg_year <- train_set %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
    left_join(reg_genre, by = "genres") %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - mu - bi - bu - b_g)/(lam + n()))

  reg_month <- train_set %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
    left_join(reg_genre, by = "genres") %>%
    left_join(reg_year, by = "year") %>%
    group_by(month) %>%
    summarize(b_m = sum(rating - mu - bi - bu - b_g - b_y)/(lam + n()))

  predictions <- test_set %>%
    left_join(reg_movie, by = "movieId") %>%
    left_join(reg_user, by = "userId") %>%
    left_join(reg_genre, by = "genres") %>%
    left_join(reg_year, by = "year") %>%
    left_join(reg_month, by = "month") %>%
    mutate(preds = mu + bi + bu + b_g + b_y + b_m) %>%
    pull(preds)

  return(RMSE(test_set$rating, predictions))
})

lambda <- lambdas[which.min(rmses_cv5)]
```

Through this cross-validation, we see that $\lambda = 4.5$ is optimal for this model. Next we apply regularization of movie, user, genre, year, and month effects using $\lambda = 4.5$.

```
lambda <- lambdas[which.min(rmses_cv5)] #this lambda is also 4.5

reg_movie <- train_set %>%
  group_by(movieId) %>%
  summarize(bi = sum(rating - mu)/(lambda + n())) #reg_movie will contain bi

reg_user <- train_set %>%
  left_join(reg_movie, by = "movieId") %>% #need to join so that we have access to bi
  group_by(userId) %>%
  summarize(bu = sum(rating - mu - bi)/(lambda + n()))

reg_genre <- train_set %>%
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - bi - bu)/(lambda + n()))

reg_year <- train_set %>%
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  left_join(reg_genre, by = "genres") %>%
  group_by(year) %>%
  summarize(b_y = sum(rating - mu - bi - bu - b_g)/(lambda + n()))
```

```
reg_month <- train_set %>%
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  left_join(reg_genre, by = "genres") %>%
  left_join(reg_year, by = "year") %>%
  group_by(month) %>%
  summarize(b_m = sum(rating - mu - bi - bu - b_g - b_y)/(lambda + n()))

pred8 <- test_set %>%
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  left_join(reg_genre, by = "genres") %>%
  left_join(reg_year, by = "year") %>%
  left_join(reg_month, by = "month") %>%
  mutate(preds = mu + bi + bu + b_g + b_y + b_m) %>%
  pull(preds)

rmse8 <- RMSE(test_set$rating, pred8)
rmse8
```

```
## [1] 0.8651888
```

This model achieved a RMSE of 0.8651888. This is the best model so far and I will apply this model to the validation set.

Results

I will be using the model that involves regularized movie + user + genres + year + month effects onto the validation set. First, we prepare the edx and validation sets by adding their month and year columns.

```
edx <- edx %>%
  mutate(year = year(as_datetime(timestamp)), month = month(as_datetime(timestamp)))

validation <- validation %>%
  mutate(year = year(as_datetime(timestamp)), month = month(as_datetime(timestamp)))
```

We will use $\lambda = 4.5$ in this model since that is the optimal λ for the regularized movie + user + genres + year + month effects model.

```
reg_movie <- edx %>%
  group_by(movieId) %>%
  summarize(bi = sum(rating - mu)/(lambda + n())) #reg_movie will contain bi

reg_user <- edx %>%
  left_join(reg_movie, by = "movieId") %>% #need to join so that we have access to bi
  group_by(userId) %>%
  summarize(bu = sum(rating - mu - bi)/(lambda + n()))

reg_genre <- edx %>%
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  group_by(genres) %>%
  summarize(b_g = sum(rating - mu - bi - bu)/(lambda + n()))

reg_year <- edx %>%
```

```r
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  left_join(reg_genre, by = "genres") %>%
  group_by(year) %>%
  summarize(b_y = sum(rating - mu - bi - bu - b_g)/(lambda + n()))

reg_month <- edx %>%
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  left_join(reg_genre, by = "genres") %>%
  left_join(reg_year, by = "year") %>%
  group_by(month) %>%
  summarize(b_m = sum(rating - mu - bi - bu - b_g - b_y)/(lambda + n()))

pred_final <- validation %>%
  left_join(reg_movie, by = "movieId") %>%
  left_join(reg_user, by = "userId") %>%
  left_join(reg_genre, by = "genres") %>%
  left_join(reg_year, by = "year") %>%
  left_join(reg_month, by = "month") %>%
  mutate(preds = mu + bi + bu + b_g + b_y + b_m) %>%
  pull(preds)

final_rmse <- RMSE(validation$rating, pred_final)
final_rmse
```

```
## [1] 0.8644099
```

Using this model, we obtain the lowest RMSE: 0.8644099!

Conclusion

To reiterate, the most effective and accurate model was the model that Regularized the movie, user, genre, year, and month effects. Using this model resulted in a RMSE of 0.8644099 when applying it to the validation set.

This project can be extended by providing recommendations to users. This would require finding movies that a user did not rate, filtering through the selected movies and choosing only ratings of either 4 or 5, and filtering by the user's most watched genres.