Capstone MovieLens

David Delfuoco

2024-10-21

Introduction

For this project I will be creating a movie recommendation system using the 10M version of the MovieLens dataset. This dataset was partitioned into a 90% to 10% split by the code provided from the course. The edx represents 90% training set and final_holdout_test represents the 10% testing set. The final_holdout_test dataset will only be used once at the very end to evaluate my model and can not be used for any of the iterative training or testing of my model. My model will attempt to predict the correct rating based on other parameters in the data.

The metric for how my model will be judged is the root mean squared error (RMSE). This is a widely used type of loss function, where the closer to zero the RMSE is the better it is performing.

Recommendation systems like this are useful to businesses in order to keep their customers happy, by suggesting products the customer should enjoy consuming or buying. This directly results to increased revenue.

My final approach was following the "Building the Recommendation System" and "Regularization" from class as the base and adding on to it. This approach uses estimates of the parameters to make a prediction, and applying a regularization effect to appropriate parts. I also applied restrictions on the predictions based on observations of what the prediction should be.

Exploring the data

I will begin by exploring the basics of the dataset to find out the overall size, structure, and find out exactly what each row and column represents. Since I know the created edx is a MovieLens dataset, I can conveniently get the definitions for each column inside of R.

These are the definitions:

Columns	Definitions
userId	Unique ID for the user.
movieId	Unique ID for the movie.
rating	A rating between 0 and 5, for the movie.
timestamp	Date and time the rating was given.
title	Movie title (not unique).
genres	Genres associated with the movie.

Now that I know what each column represents, I'll find out what each row represents. I find the best way to do this is to just look at the first few rows, and it should become apparent.

```
head(edx)
```

```
##   userId movieId rating timestamp                          title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                           genres
## 1                Comedy|Romance
## 2          Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5        Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7       Children|Comedy|Fantasy
```

From this I can see that each row of edx represents a single rating of a movie by one user. Let's check the size of the edx dataset.

```
dim(edx)
```

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

Well the name of the original dataset being the 10M version of MovieLens makes sense. Since edx is a 90% partition of that dataset, we can see that it has a little over 9 million rows and 6 columns. The sheer size of the dataset will make most prebuilt machine learning algorithms that come with R have a very long runtime on my computers limited hardware. I will have to explore ways to make the dataset smaller by either taking a much smaller subset of the data, or using mathematical estimations to represent the data.

Let's now check what each of the columns datatypes are. This is important to know, as this will tell you what types of functions you can perform on them, and any extra modifications you to need to perform on the data to use a particular function. We can explore the overall structure like this.

```
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  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|A
```

Most of this information we already explored, but what we care about in this are the data types. We can see that the userId, movieId, and timestamp are integers. Title and genres are characters, lastly rating is numeric. It's important to note that these aren't the true representations of what these columns are, and are most likely in this data type for efficiency and size reduction. UserId, movieId, and genres are factors aka categorical parameters. Genres is special in that each different genre combination that is associated

with a movie would be considered a new category. Since we know timestamp represents the date and time the movie was rated, it is most likely the time in seconds from the epoch.

Lastly lets explore how many different values for each column, and the possible range of the rating column. The rating column had a range of 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5. This shows that the actual rating range is 0.5-5.0, and only changes in increments of 0.5. Here is a table for counting the unique amount for each column:

Columns	Count
userId	69878
movieId	10677
rating	10
timestamp	6519590
title	10676
genres	797

We already showed that rating can only have 10 different values, but out of these 9 million ratings, there were 69 thousand users, 10 thousand movies, and 797 different combination of genres. Unsurprisingly the most unique column is timestamp as it is very unlikely for multiple users to rate a movie at the exact same second in time.

Getting a Dataset for Testing

Since I can't use the final_holdout_test for any iterative testing, I feel like this would be a good time to partition the edx dataset into a new training and testing set. This is a necessary step as you don't want to test your model on the same data that was used to train it, as this will cause overtraining and not perform well on outside data. I will be using the exact same method the edx and final_holdout_test was created from the provided code, and will also create a 90% training and 10% testing set from the edx dataset. To make it clear how it was created and what seed I used I will provide the code here.

```
# The method for creating the partitions were taken from the given code
# creating the edx and final_holdout_test datasets

# Partition the edx dataset for refining and avoid overtraining
set.seed(5)
edx_test_index <- createDataPartition(y = edx$rating, times = 1,
                                      p = 0.1, list = FALSE)
tmp <- edx[edx_test_index,]
edx_train <- edx[-edx_test_index,]

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

# Add rows removed from edx_test set back into edx_train set
removed <- anti_join(tmp, edx_test)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

```
edx_train <- rbind(edx_train, removed)
```

To be clear I will be doing all of my training on the edx_train dataset and all of my testing on the edx_test dataset to help improve my model. Just to make sure that there is no missing data from my new datasets, I will check for any na values.

```
# Check for any na
any(is.na(edx_train))
```

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

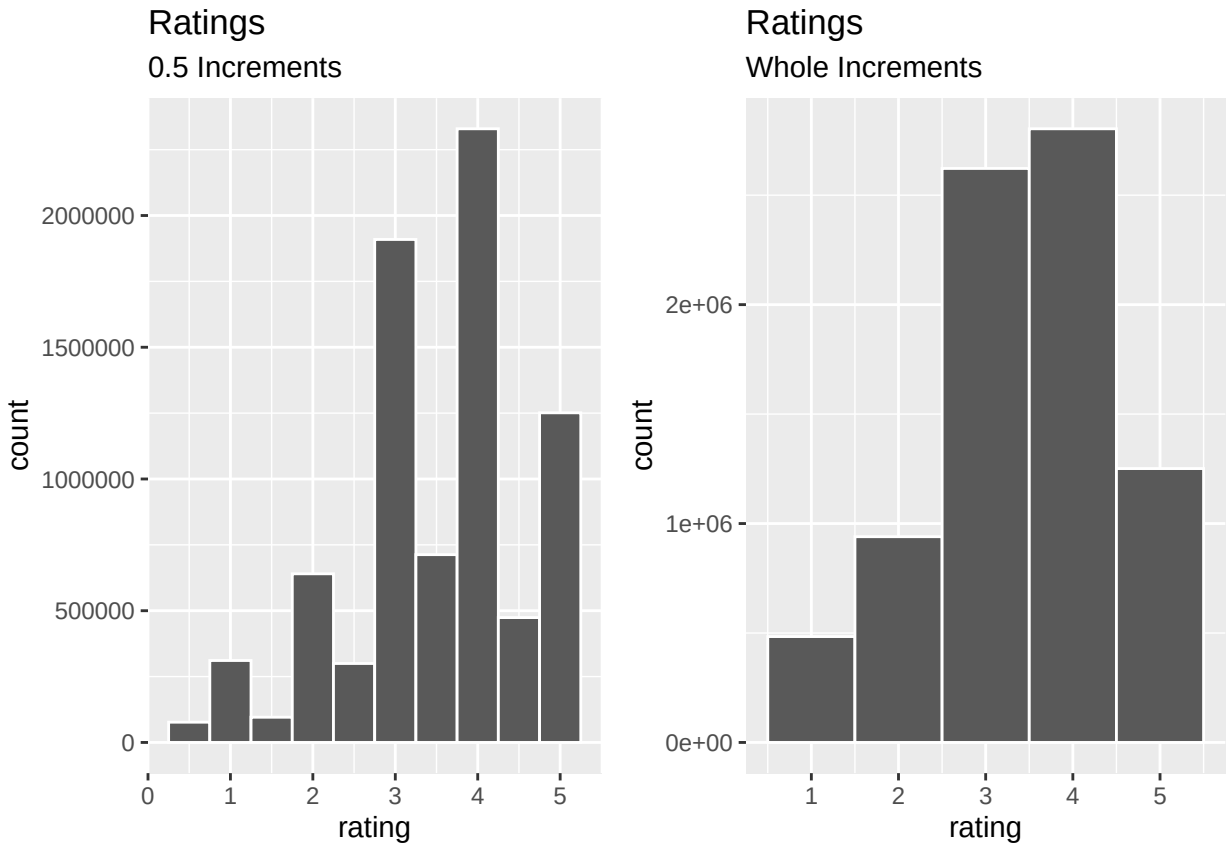
```
any(is.na(edx_test))
```

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

Since there are no missing values we can move on.

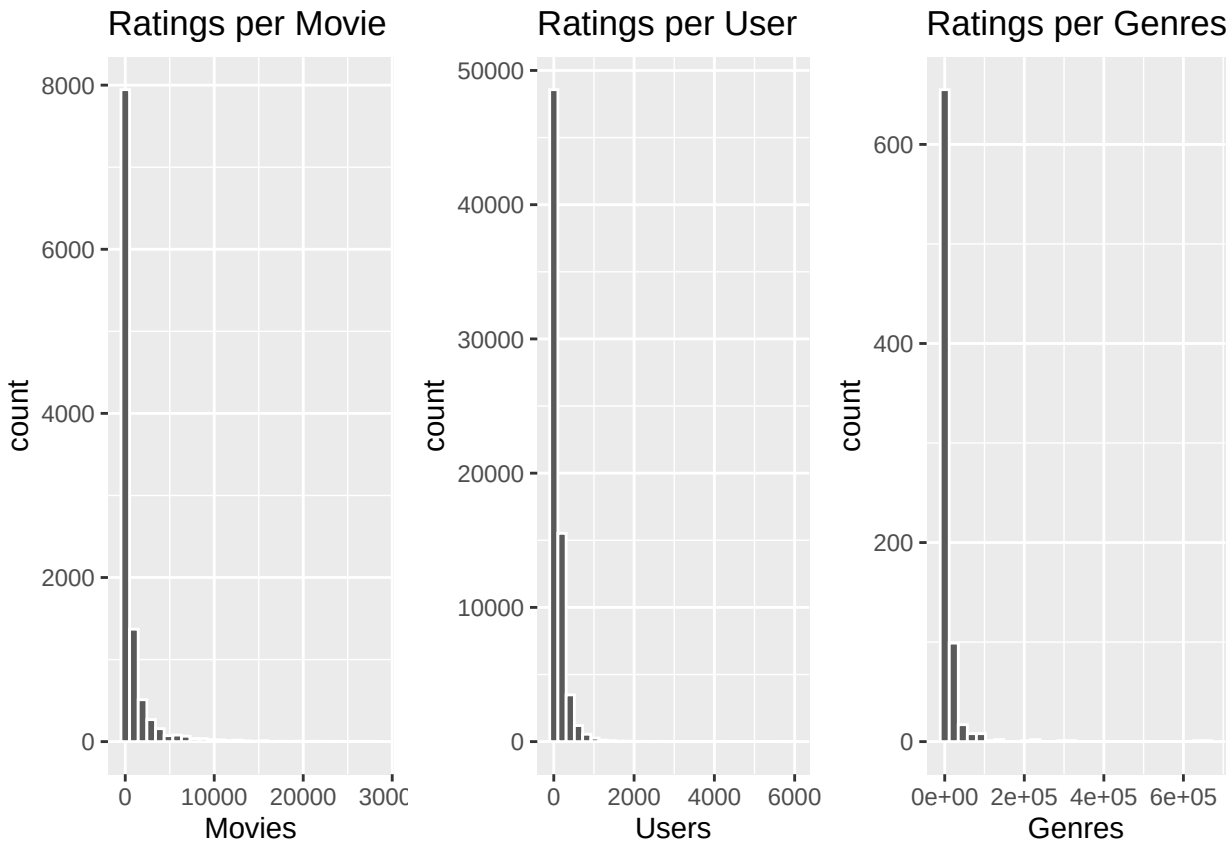
Visual Exploration

Let's see what type of information we can gather by visualizing the data, I will be using the new edx_train dataset for all of these graphs. Let's make histograms for our columns and see if any patterns arise. We'll start with the rating column.

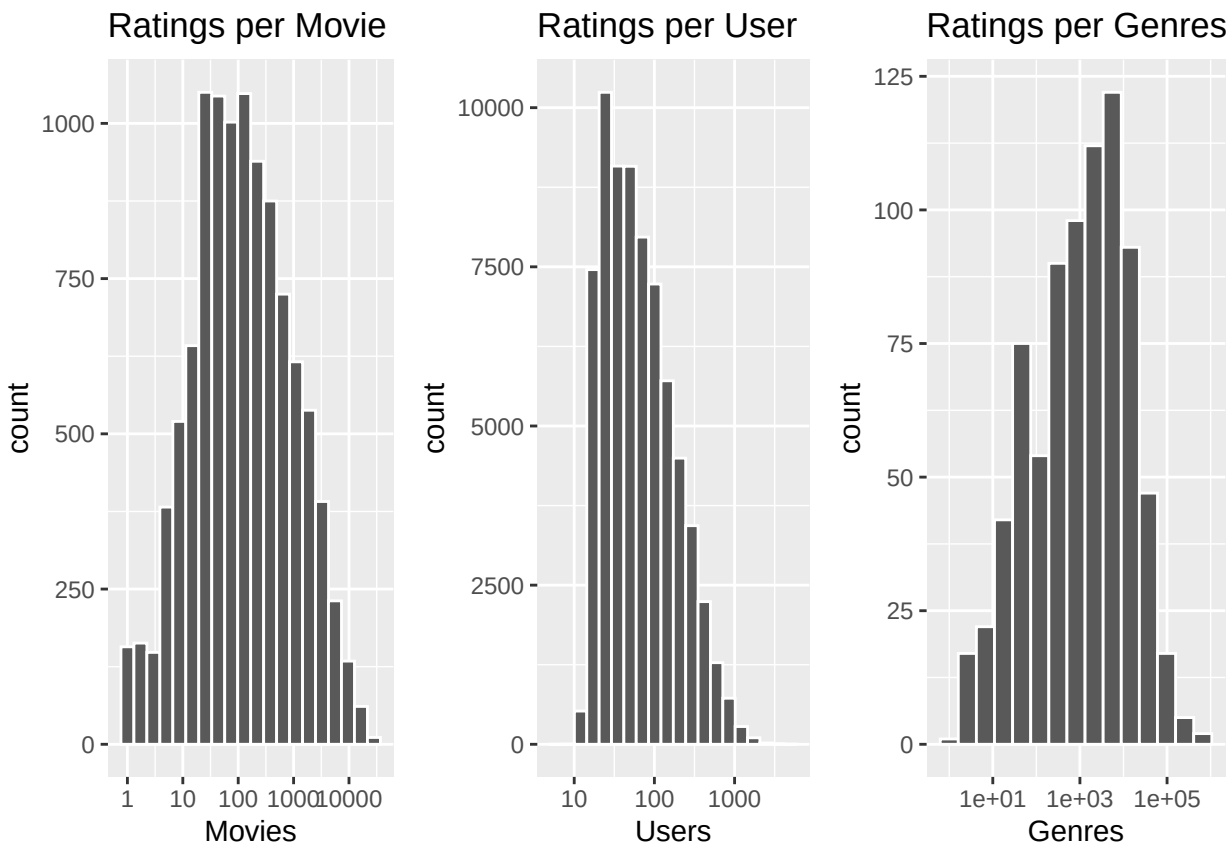


We can see that the majority of users rated a movie between 3 and 4, and that it is a lot more likely that a user will rate with a whole number.

Now let's look at histograms of the movies, users, and genres:



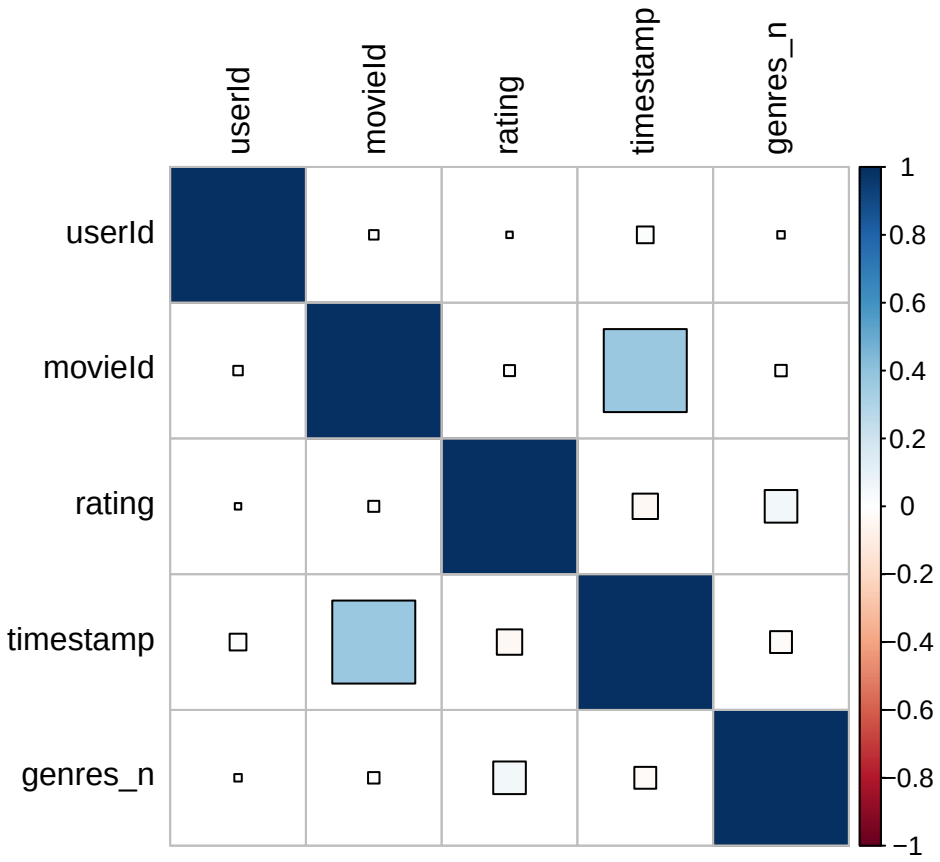
These columns are all right-skewed, showing that some movies, users, and genres get a lot more ratings then the rest. Let's use the scale_x_log10() function on them to get a better view with out the heavily rated sections influencing the rest of the graph.



These are now closer to a normal distribution, but we can see that users are still being skewed even after being logarithmically scaled.

Lastly let's explore the correlation of these columns. If one predictor column is heavily correlated with another predictor then it won't be suitable for making predictions until you can account for that correlation. Since I want the correlation of the genres column, I need to convert its information into a numeric column. The easiest method I can think of is factoring the genres column and numbering them by each factor.

```
# Number the genres to get the correlation
edx_train <- edx_train %>% mutate(genres_n = as.numeric(factor(genres)))

# Compute correlation of numeric columns
cors <- edx_train %>% select(-c(title,genres)) %>% cor()
# Plot the correlation using the corrplot package
corrplot(cors, method = "square", outline = T, tl.col = "black")
```



This shows that no columns are heavily correlated. Surprisingly the strongest correlation is between movieId and timestamp. Perhaps this correlation is related to a movies new release and people want to see it right when it comes out and rate it, or a particular movie is currently buzzworthy getting the same treatment of being rated in a small timeframe.

Let's check if this correlation is statistically significant:

```
# Check p-value of the move correlated predictors
cor.test(edx$movieId, edx$timestamp)
```

```
##
##  Pearson's product-moment correlation
##
## data:  edx$movieId and edx$timestamp
## t = 1209.5, df = 9000053, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.3733661 0.3744900
## sample estimates:
##       cor
## 0.3739282
```

We see that the p-value is $< 2.2 * 10^{-16}$, showing that it is indeed significant. For this reason I will not be exploring the timestamp column as a predictor.

Constructing the Models

Since the dataset is so large, I decided to use estimations to construct the training model. I did try to use the train() function from the caret package, but even at taking a subset of 10,000 from the edx_train, my runtimes were very large or run out memory. If I was having so many problems at 10,000 I decided that to get anywhere close to using a train() would be a decent loss in data. Before I get into the estimations to construct the model, I'm going to create our RMSE function, the metric in which I judge how well the model is doing. RMSE is represented as the value we are predicting, so in this case an improvement of 0.5 in the RMSE means the "the typical error we make when predicting a movie rating"[1] will be 0.5 closer to the true value. The mathematical representation of RMSE is:

$$\sqrt{\frac{1}{N}\sum(y-\hat{y})^2}$$

Where N is the number of ratings, y is the true rating, and $\hat{y_i}$ is the predicted rating. This is the RMSE function in R.

```
# Create the rmse function
rmse <- function(true_rating, predicted_rating) {
  sqrt(mean((true_rating - predicted_rating)^2))
}
```

I'm going to build my model by following the formula:

$$Y = \mu + b$$

Where Y is the rating, μ is the average of the rating, each b will be the effect of a predictor. Since these represent the true values from the training data, we will be using the hat of these variables which represent the estimation of all ratings to make our prediction. We represent $\hat{\mu}$ as the average of all ratings, and $\hat{Y}$ as the predicted ratings. Doing this we can get the $\hat{b}$, the estimation of all predictor effects by getting the average of $Y - \hat{\mu}$. Taking the average of the sample is an efficient way to get an estimation. Following this logic we can slowly add each effect from columns until the model is good enough.

Constructing the Base Model

Starting with calculating $\hat{\mu}$:

```
# Following the method from "Building the Recommendation System"
# from class, the code follows very close building the rmse to
# getting the predictions for the movie + user effects model (up to bu_hat)

# Create mu_hat
mu_hat <- mean(edx_train$rating)

# naive rmse
naive_rmse <- rmse(edx_test$rating, mu_hat)
naive_rmse
```

```
## [1] 1.060119
```

So the starting RMSE is 1.0601188, the goal now is to iteratively adjust this model, and by lowering the RMSE we know the model is getting better.

Now lets add $\hat{b_i}$ an estimation of the movie effects on the model.

```
# Create the bi_hat values (movie effects)
movie_avgs <- edx_train %>%
  group_by(movieId) %>%
  summarise(bi_hat = mean(rating - mu_hat))

# Prediction values based on movie effects model on edx_test
pred_movie <- edx_test %>%
  left_join(movie_avgs, by = "movieId") %>%
  mutate(preds = mu_hat + bi_hat) %>%
  pull(preds)

# RMSE for movie effects model
movie_rmse <- rmse(edx_test$rating, pred_movie)
movie_rmse
```

```
## [1] 0.9439196
```

We see another improvement but any number added to the average would be an improvement. Let's add the user effects to see if it still improves.

Adding the $\hat{b_u}$ an estimation of the user effects on the model.

```
# Create the bu_hat values (user effects)
user_avgs <- edx_train %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarise(bu_hat = mean(rating - mu_hat - bi_hat))

# Prediction values based on movie effects + user effects model on edx_test
pred_m_u <- edx_test %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(preds = mu_hat + bi_hat + bu_hat) %>%
  mutate(preds = preds) %>%
  pull(preds)

# RMSE for movie + user effects model
m_u_rmse <- rmse(edx_test$rating, pred_m_u)
m_u_rmse
```

```
## [1] 0.8664372
```

Another improvement in the RMSE for the model, this time by 0.0774823

The base models so far:

Methods	RMSE
Just the average	1.0601188

Methods	RMSE
Movie Effects Model	0.9439196
Movie + User Effects Model	0.8664372

Testing Clamping and Rounding

Before I add another effect to the model, I want to test if clamping and rounding the predictions improves the RMSE. The logic behind this is from our exploration of the rating values, we know that the range if from 0.5-5.0 so I can clamp any predictions to this range. Since ratings our only given in 0.5 increments I can round the predictions to the nearest 0.5 and see if that improves the RMSE.

Since I couldn't find a base R function for clamp I decided to make the function for this particular case.

```
# Testing Clamping
# Make a function to clamp to 0.5-5
clamp <- (function(preds){
  p_clamp <- sapply(preds, function(p){
    if(p < 0.5) {
      return(0.5)
    } else if (p > 5 ) {
      return(5)
    } else {
      return(p)
    }
  })
  return(p_clamp)
})

# Testing clamp on earlier predictions
# Clamping the predictions from movie effects model
movie_rmse_c <- rmse(edx_test$rating, clamp(pred_movie))
movie_rmse_c
```

```
## [1] 0.9439196
```

```
movie_rmse # unclamped rmse of movie effects model
```

```
## [1] 0.9439196
```

```
# Clamping the predictions from movie + user effects model
m_u_rmse_c <- rmse(edx_test$rating, clamp(pred_m_u))
m_u_rmse_c
```

```
## [1] 0.866228
```

```
m_u_rmse # unclamped rmse of movie + user effects model
```

```
## [1] 0.8664372
```

The movie effects model didn't improve but the movie + user effect model did. Let's check the minimum and maximum of these predictions to see if this can be explained. The minimum of pred_movie is 0.5 and maximum is 5. The movie effects predictions stayed within the range so clamp did nothing which is why the RMSE didn't change. The minimum of pred_m_u is -0.284 and maximum is 6.08. The
movie+user effects predictions did leave the range so clamp caught those outliers and brought them in range to improve the RMSE. So clamping predictions will be a new permanent step since it seems to only help the RMSE.

There is a function to round the nearest 0.5, but this function round_any() comes from the plyr package and was causing problems with the dplyr package. For this reason I will also test this manually instead of using a built in function.

```
# Testing rounding to the nearest 0.5 on earlier predictions
# Rounding the predictions from movie effects model
movie_rmse_r <- rmse(edx_test$rating, round(pred_movie * 2) / 2)
movie_rmse_r
```

```
## [1] 0.9548239
```

```
movie_rmse # non rounded rmse of movie effects model
```

```
## [1] 0.9439196
```

```
# Rounding the predictions from movie + user effects model
m_u_rmse_r <- rmse(edx_test$rating, round(pred_m_u * 2) / 2)
m_u_rmse_r
```

```
## [1] 0.8783042
```

```
m_u_rmse # non rounded rmse of movie + user effects model
```

```
## [1] 0.8664372
```

Rounding seems to degrade the RMSE in both cases. I assume that since the RMSE takes the mean of the difference between true and predicted ratings, this rounding actually pushes the mean of the predicted farther away overall from the mean of the true values.

Rounding did not improve the RMSE but clamping did. This was a small improvement to the RMSE but still an improvement.

Table of the model so far:

Methods	RMSE
Movie Effects Model	0.9439196
Movie Effects Model Clamped	0.9439196
Movie Effects Model Rounded	0.9548239
Movie + User Effects Model	0.8664372
Movie + User Effects Model Clamped	0.8662280
Movie + User Effects Model Rounded	0.8783042

Adding Genres Effect and Regularization to Model

Let's add the genres effect to the best model so far, we will continue with the same logic as before for computing this new $\hat{b_g}$.

```
# Applying the same method from earlier to add another effect to the model

# Create the bg_hat values (genres effects)
genre_avg <- edx_train %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  group_by(genres) %>%
  summarise(bg_hat = mean(rating - mu_hat - bi_hat - bu_hat))

# Prediction values based on movie + user + genres effect model on edx_test
pred_m_u_g <- edx_test %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(genre_avg, by = "genres") %>%
  mutate(preds = mu_hat + bi_hat + bu_hat + bg_hat) %>%
  mutate(preds = clamp(preds)) %>%
  pull(preds)

# RMSE for movie + user + genres effects model
m_u_g_rmse <- rmse(edx_test$rating, pred_m_u_g)
m_u_g_rmse
```

```
## [1] 0.8658639
```

```
m_u_rmse_c # earlier best model
```

```
## [1] 0.866228
```

The model is still improving but the improvements are getting smaller, this time it only improved by 3.64×10^{-4}.

I'm not going to add the effects from titles columns and timestamp, so now the last improvement to the model I will try is regularization. Earlier we saw how the histograms of the non-scaled userId, movieId, and genres were heavily right-skewed. Since our estimates of these effects are the mean, these highly rated movies, users that rate a lot, and genres that get rated more can effect the mean by not being a good representation of the highly rated and lowly rated effects. Regularization solves this problem by "permitting us to penalize large estimates that are formed using small sample sizes."[2]. The implementation I use follows the regularization course where this is our current best model:

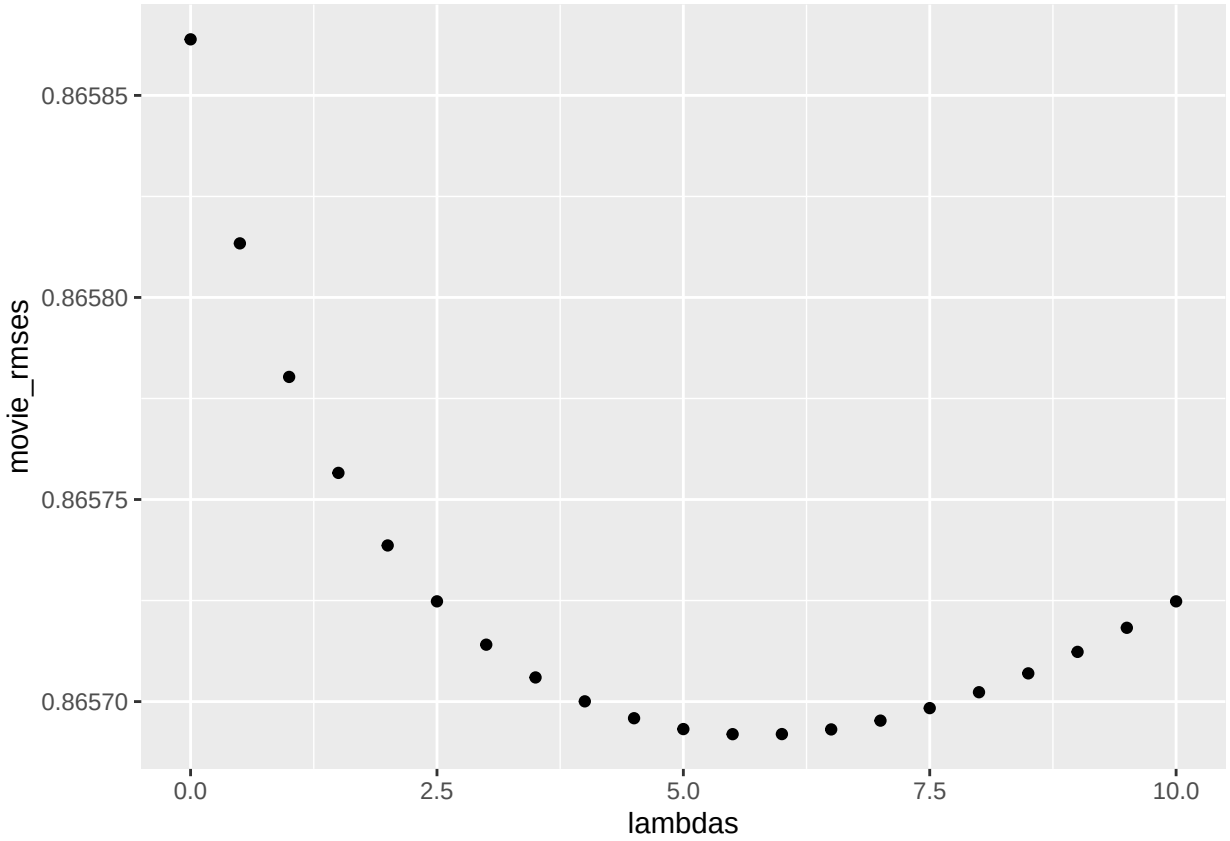
$$\hat{Y} = \hat{\mu} + \hat{b_i} + \hat{b_u} + \hat{b_g}$$

To implement the regularization of $\hat{b_i}$, we no longer compute it by $\frac{1}{n}\sum(y - \hat{\mu})$. It is now computed by $\hat{b_i}(\lambda) = \frac{1}{\lambda + n_i}\sum(y - \hat{\mu})$
Where n_i is the number of ratings movie i, and λ is a tuning parameter where the larger λ is the more it shrinks.[2]

Since λ is a tuning parameter I will run a vector of λ's through a sapply() loop to find the one that gives the best RMSE.

```
# Following the method from "Regularization"
# from class, the code follows very close from regularizing
# bi_h and bu_h and using the same method to apply it to bg_h

# Lambdas to test on regularizing
lambdas <- seq(0, 10, 0.5)

# Regularization of movies on full effects model
movie_rmses <- sapply(lambdas, function(l){

  mu_hat <- mean(edx_train$rating)
  # Regularized movies
  bi_h <- edx_train %>%
    group_by(movieId) %>%
    summarise(bi_h = sum(rating - mu_hat) / (n() + l))

  bu_h <- edx_train %>%
    left_join(bi_h, by = "movieId") %>%
    group_by(userId) %>%
    summarise(bu_h = mean(rating - mu_hat - bi_h))

  bg_h <- edx_train %>%
    left_join(bi_h, by = "movieId") %>%
    left_join(bu_h, by = "userId") %>%
    group_by(genres) %>%
    summarise(bg_h = mean(rating - mu_hat - bi_h - bu_h))

  pred_r <- edx_test %>%
    left_join(bi_h, by = "movieId") %>%
    left_join(bu_h, by = "userId") %>%
    left_join(bg_h, by = "genres") %>%
    mutate(pred = mu_hat + bi_h + bu_h + bg_h) %>%
    mutate(pred = clamp(pred)) %>%
    pull(pred)

  return(rmse(edx_test$rating, pred_r))
})
```

We can see how λ effects the RMSE with this plot:



We see the best lambda is a little greater than 5 and best RMSE is a little less than .8657. Let's get the exact numbers to check and save the lambda in case this is the best model to be used on the final RMSE check.

```
# Best lambda for regularized movies on full effects model
movie_lambda <- lambdas[which.min(movie_rmses)]
movie_lambda
```

```
## [1] 5.5
```

```
# RMSE for regularized movies on full effects model
movie_reg <- min(movie_rmses)
movie_reg
```

```
## [1] 0.8656919
```

```
m_u_g_rmse # earlier best model
```

```
## [1] 0.8658639
```

We see the RMSE did improve but the improvements are getting smaller and smaller. This time we only get an improvement of 1.72×10^{-4}. Let's try regularizing the other effects and see if the model keeps improving.

Now let's regularize the movies and users using the same methods.

```
# Regularization of movies and users on full effects model
m_u_rmses <- sapply(lambdas, function(l){

  mu_hat <- mean(edx_train$rating)
  # Regularized movies
  bi_h <- edx_train %>%
    group_by(movieId) %>%
    summarise(bi_h = sum(rating - mu_hat) / (n() + l))
  # Regularized  users
  bu_h <- edx_train %>%
    left_join(bi_h, by = "movieId") %>%
    group_by(userId) %>%
    summarise(bu_h = sum(rating - mu_hat - bi_h) / (n() + l))

  bg_h <- edx_train %>%
    left_join(bi_h, by = "movieId") %>%
    left_join(bu_h, by = "userId") %>%
    group_by(genres) %>%
    summarise(bg_h = mean(rating - mu_hat - bi_h - bu_h))

  pred_r <- edx_test %>%
    left_join(bi_h, by = "movieId") %>%
    left_join(bu_h, by = "userId") %>%
    left_join(bg_h, by = "genres") %>%
    mutate(pred = mu_hat + bi_h + bu_h + bg_h) %>%
    mutate(pred = clamp(pred)) %>%
    pull(pred)

  return(rmse(edx_test$rating, pred_r))
})

# Best lambda for full effects model when regularizing movies and users
m_u_lambda <- lambdas[which.min(m_u_rmses)]
m_u_lambda
```

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

```
# RMSE for full effects model when regularizing movies and users
m_u_reg <- min(m_u_rmses)
m_u_reg
```

```
## [1] 0.8653994
```

```
movie_reg # earlier best model
```

```
## [1] 0.8656919
```

The model is still improving, this time by a slightly larger margin of 2.93×10^{-4}. We can also see that the best λ changed for this model which is why it's
important to loop through the λ's to find the best fit.

Now let's try regularizing movies, user, and genres using the same methods.

```r
# Regularization of movies, users, and genres on full effects model
m_u_g_rmses <- sapply(lambdas, function(l){

  mu_hat <- mean(edx_train$rating)
  # Regularized movies
  bi_h <- edx_train %>%
    group_by(movieId) %>%
    summarise(bi_h = sum(rating - mu_hat) / (n() + l))
  # Regularized users
  bu_h <- edx_train %>%
    left_join(bi_h, by = "movieId") %>%
    group_by(userId) %>%
    summarise(bu_h = sum(rating - mu_hat - bi_h) / (n() + l))
  # Regularized genres
  bg_h <- edx_train %>%
    left_join(bi_h, by = "movieId") %>%
    left_join(bu_h, by = "userId") %>%
    group_by(genres) %>%
    summarise(bg_h = sum(rating - mu_hat - bi_h - bu_h) / (n() + l))

  pred_r <- edx_test %>%
    left_join(bi_h, by = "movieId") %>%
    left_join(bu_h, by = "userId") %>%
    left_join(bg_h, by = "genres") %>%
    mutate(pred = mu_hat + bi_h + bu_h + bg_h) %>%
    mutate(pred = clamp(pred)) %>%
    pull(pred)

  return(rmse(edx_test$rating, pred_r))
})

# Best lambda for model when regularizing movies, users, and genres
m_u_g_lambda <- lambdas[which.min(movie_rmses)]
m_u_g_lambda
```

```
## [1] 5.5
```

```r
# RMSE for full effects model when regularizing movies, users, and genres
m_u_g_reg <- min(m_u_g_rmses)
m_u_g_reg
```

```
## [1] 0.8653999
```

```r
m_u_reg # earlier best model
```

```
## [1] 0.8653994
```

This actually hurt the RMSE by a very small amount. It looks like it's better to not regularize the genres effect.

Lastly I'm going to try to fine tune the best $\lambda(4.5)$ on our best model so far to see if there is a better one. The first set of λ's I used incremented

by 0.5 to save on testing time. I will now try in increments of 0.1 around the best λ to see if one exists between the unchecked margins.

```
# Try to fine tune lambda for a better result on our best model
lambdas_fine <- seq(4.1, 4.9, 0.1)
m_u_rmses <- sapply(lambdas_fine, function(l){

  mu_hat <- mean(edx_train$rating)
  # REgularized movies
  bi_h <- edx_train %>%
    group_by(movieId) %>%
    summarise(bi_h = sum(rating - mu_hat) / (n() + l))
  # Regularized users
  bu_h <- edx_train %>%
    left_join(bi_h, by = "movieId") %>%
    group_by(userId) %>%
    summarise(bu_h = sum(rating - mu_hat - bi_h) / (n() + l))

  bg_h <- edx_train %>%
    left_join(bi_h, by = "movieId") %>%
    left_join(bu_h, by = "userId") %>%
    group_by(genres) %>%
    summarise(bg_h = mean(rating - mu_hat - bi_h - bu_h))

  pred_r <- edx_test %>%
    left_join(bi_h, by = "movieId") %>%
    left_join(bu_h, by = "userId") %>%
    left_join(bg_h, by = "genres") %>%
    mutate(pred = mu_hat + bi_h + bu_h + bg_h) %>%
    mutate(pred = clamp(pred)) %>%
    pull(pred)

  return(rmse(edx_test$rating, pred_r))
})

# Find out if a better lambda was found
m_u_lambda_fine <- lambdas_fine[which.min(m_u_rmses)]
m_u_lambda_fine
```

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

```
m_u_lambda # earlier best lambda
```

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

The best λ didn't change so this is the one that will be used with the best model.

The models and their RMSE for this section are:

Methods	RMSE
Full Effects Model	0.8658639
Regularized Movie on Full Effects Model	0.8656919

Methods	RMSE
Regularized Movie + User on Full Effects Model	0.8653994
Regularized Movie + User + Genres Model	0.8653999

The improvements are getting quite small now so I'm going to choose the regularized movie and user on full effects model as the final model and test it on the final_holdout_test.

Results

Finally we get to test the best model on the final_holdout_test to see how it compares to some data that wasn't part of the edx dataset. Up until now all the test were done on the edx_test dataset which was a partition of the edx. We construct the full model from the regularized movies + users on the full effects model with the best λ and clamping on the predictions to see how we do.

```
# Create our final model to get predictions from final_holdout_test
lambda_final <- m_u_lambda_fine
mu_hat <- mean(edx_train$rating)
# Regularized
bi_h <- edx_train %>%
  group_by(movieId) %>%
  summarise(bi_h = sum(rating - mu_hat) / (n() + lambda_final))
# Regularized
bu_h <- edx_train %>%
  left_join(bi_h, by = "movieId") %>%
  group_by(userId) %>%
  summarise(bu_h = sum(rating - mu_hat - bi_h) / (n() + lambda_final))

bg_h <- edx_train %>%
  left_join(bi_h, by = "movieId") %>%
  left_join(bu_h, by = "userId") %>%
  group_by(genres) %>%
  summarise(bg_h = mean(rating - mu_hat - bi_h - bu_h))

# Get predictions of the final model on final_holdout_test
final_pred <- final_holdout_test %>%
  left_join(bi_h, by = "movieId") %>%
  left_join(bu_h, by = "userId") %>%
  left_join(bg_h, by = "genres") %>%
  mutate(pred = mu_hat + bi_h + bu_h + bg_h) %>%
  mutate(clamp(pred)) %>%
  pull(pred)

# RMSE for our final model on final_holdout_test
final_rmse <- rmse(final_holdout_test$rating, final_pred)
final_rmse
```

```
## [1] 0.8648799
```

Surprisingly the model performed better on the outside data then it did on the edx_test data. Generally the model usually performs slightly worse on the outside data.

This result could be from making sure to avoid overtraining or just luck. The end models predictions were on average 0.8648799 off from the actual final_holdout_test ratings.

A representation of the final model is:

$$\hat{Y} = \hat{\mu} + \hat{b_i}(\lambda) + \hat{b_u}(\lambda) + \hat{b_g}$$
$$\hat{b_i}(\lambda) = \frac{1}{\lambda + n_i} \sum (Y - \hat{\mu})$$
$$\hat{b_u}(\lambda) = \frac{1}{\lambda + n_u} \sum (Y - \hat{\mu} - \hat{b_i}(\lambda))$$

Where $\hat{Y}$ is the predicted rating, $\hat{\mu}$ is the estimation of the average rating, $\hat{b_i}$ is the estimated movie effect regulated by λ, $\hat{b_u}$ is the estimated user effect regulated by λ, and $\hat{b_g}$ is the estimated genres effect.

A final rundown of all the methods and their RMSE values:

Methods	RMSE
Movie Effects Model	0.9439196
Movie Effects Model Clamped	0.9439196
Movie Effects Model Rounded	0.9548239
Movie + User Effects Model	0.8664372
Movie + User Effects Model Clamped	0.8662280
Movie + User Effects Model Rounded	0.8783042
Full Effects Model	0.8658639
Regularized Movie on Full Effects Model	0.8656919
Regularized Movie + User on Full Effects Model	0.8653994
Regularized Movie + User + Genres Model	0.8653999
Best Model on final_holdout_test	0.8648799

Note all models after "Movie + User Effect Model Rounded" are clamped.

Conclusion

Given more time I would of tried to tackle the timestamp parameter and see how seperating this column into appropriate year, month, day columns to see how that would effect the model. With better computing hardware, I would of also liked to bruteforce run the edx dataset through some of the methods of the train() function in the caret package to see how good its RMSE would be.

In the end the final model was constructed by iteratively improving it by using RMSE as a metric to track its performance. Started by adding estimations of the parameters to help predict a rating, and then add on to this model by regularizing these effects and finally clamping the predictions to the min and max of known possible outcomes of the rating. This method performed better than I anticipated reaching an RMSE of 0.8648799 on the final_holdout_test.

References

1: Irizarry, R. A. (n.d.). Introduction to data science. Chapter 33 Large datasets. https://rafalab.dfci.harvard.edu/dsbook/large-datasets.html#netflix-loss-function

2: Irizarry, R. A. (n.d.). Introduction to data science. Chapter 33 Large datasets. https://rafalab.dfci.harvard.edu/dsbook/large-datasets.html#penalized-least-squares