MovieLens Prediction Algorithm Project
HarvardX: Capstone Project - PH125.9x

Michael Vatt

04 JAN 2020

Contents

Overview

This project stemmed from the MovieLens Dataset provided by the HarvardX Data Science Capstone Course PH125.9x. The MovieLens Dataset contains data about moviegoers and movieratings. This project is outlined as follows: once the dataset is loaded, we start with familiarizing oneself with the data, perform exploratory data analysis, move into the Methods and Analysis with several plots to further interpret its essence, and then the RMSE is developed and constructed. Exploratory analysis cannot be underestimated as it is critical to develop a machine learning algorithm that is both informative and meaningful in predicting anything. The final model is then explained and hopefully successful. Here, our desire is to develop an ability to predict movie ratings.

Introduction

This machine learning algorithm is dependent upon moviegoers (hereinafter "users") that give ratings on movies they have watched. It takes their ratings and behaviors to make its own ability of specific recommendations. Most companies desire this sort of ratings feedback in order to develop recommendation systems (e.g., Netflix, Home Depot, Apple). Recommendation Systems may be represented by ways of displaying possible items that a user of a store's webpage may be interested in purchasing, or displaying possible movie titles that others with similar tastes and behaviors have enjoyed that the current user might also enjoy. In fact, many companies collect massive datasets on their customers in order to use high ratings and low ratings as indicators and warnings for advertisement and marketing strategies (i.e., targeted marketing) through recommendation algorithms.

Netflix, the Apple Store, Google, and others arguably have mastered this capability to target additional income opportunities and customer acquistion. Netflix went as far as to award anyone who could demonstrate the ability to develop a new algorithm that would improve their own recommendation system by 10% or

greater in exchange for a $1 million prize! This was done by writing an algorithm to evaluate solely previous ratings! No information about the users or films were necessary.

This project utilized the 10M version of the MovieLens Dataset from GroupLens Research with 10 million ratings applied to 10,000 movies, by 72,000 users in order to be small enough to be feasible but large enough to be instructive and enlightening. This dataset was released in January 2009. The full dataset had roughly 27 million ratings, 58,000 movies, and 280,000 users!

Objective

The objective is to create and train a machine learning algorithm to predict user ratings using a large dataset of previous user ratings as inputs ranging from 0.5 to 5 stars. This dataset was provided by the HarvardX course from the GroupLens Research project. The algorithm's predictions of movie ratings were evaluated against a validation set given by HarvardX.

The term used to give evaluation value to the algorithm is the Root Mean Square Error - also known as RMSE. This measure is a popular method for statistics, estimation, mathematics, and applications. It is used to measure the differences between values predicted by a model and the values observed.

RMSE is a measure of accuracy - without it, the algorithm is impractical and ineffective. The lower an RMSE score, the better! The goal of this project is to develop a model that will produce an RMSE below 0.865 - a lofty goal.

The effect errors have on the RMSE is proportional to the size of the squared error - larger the error, larger the effect. A means to help navigate this problem is built into the formula by reducing the impact of the errors through the square root function.

This project's machine learning algorithm was established through an incremental and iterative development process. Different models were established in layers to compare RMSE results in order to evaluate their prediction qualities. This will enable the best model to be determined and used to predict movie ratings on the dataset.

For insight and comprehension, the RMSE is computed with the following equation:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

Dataset

The following code was provided by HarvardX to be automatically downloaded, structured, and prepared equally for each student:

- The dataset is downloaded from the MovieLens 10M webpage: https://grouplens.org/datasets/movielens/10m/

```
##########################################################
# Create edx set, validation set, and submission file
##########################################################
# Note: this process could take a couple of minutes for loading required package: tidyverse and
# package caret

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")

library(knitr)

dl <- tempfile()
```

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

ratings <- read.table(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")

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")

# The Validation subset will be 10% of the MovieLens data.

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 subset:

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)
```

In order to predict movie ratings of those users that have not seen the movie yet, the dataset is split into two sets to train and then validate the algorithm separately. These sets were supplied to us through the code above from HarvardX with the training dataset "edx" and the testing dataset "validation". The development process of the models are performed solely on the edx dataset. The model testing is used against the validation dataset.

Methods & Analysis

The first step in this process is to familiarize oneself on the MovieLens Dataset. The simplest form of familiarization is performed initially and then expounded upon. Foremost, we get to see the variables that makeup the edx data frame. If the table were to be laid out with no additional formatting or manipulations, we would see that each row represents a single rating, by a single user, for single movie.

```r
head(edx) %>% kable()
```

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	231	5	838983392	Dumb & Dumber (1994)	Comedy
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi

Next, a summary function action will provide useful feedback for each of the variables, to include the range of ratings, class, and userId/movieId information.

```
summary(edx) %>% kable()
```

userId	movieId	rating	timestamp	title	genres
Min. : 1	Min. : 1	Min. :0.500	Min. :7.897e+08	Length:9000061	Length:9000061
1st Qu.:18122	1st Qu.: 648	1st Qu.:3.000	1st Qu.:9.468e+08	Class :character	Class :character
Median :35743	Median : 1834	Median :4.000	Median :1.035e+09	Mode :character	Mode :character
Mean :35869	Mean : 4120	Mean :3.512	Mean :1.033e+09	NA	NA
3rd Qu.:53602	3rd Qu.: 3624	3rd Qu.:4.000	3rd Qu.:1.127e+09	NA	NA
Max. :71567	Max. :65133	Max. :5.000	Max. :1.231e+09	NA	NA

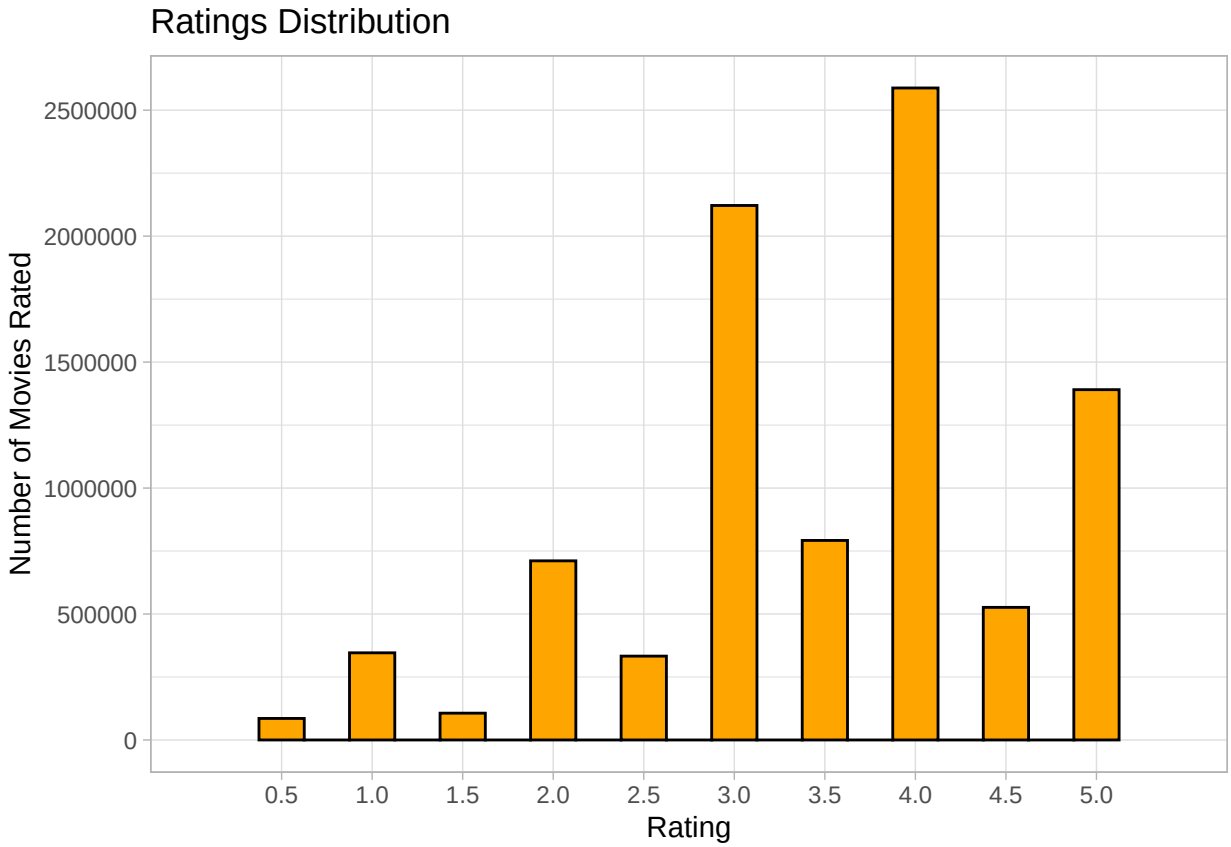
Now, we want to get an idea of how many distinct users and distinct movies there are in the dataset - resulting in approximately 70,000 users and 10,000 movies!

```
edx %>% summarize(dist_users = n_distinct(userId), dist_movies = n_distinct(movieId)) %>% kable()
```

dist_users	dist_movies
69878	10677

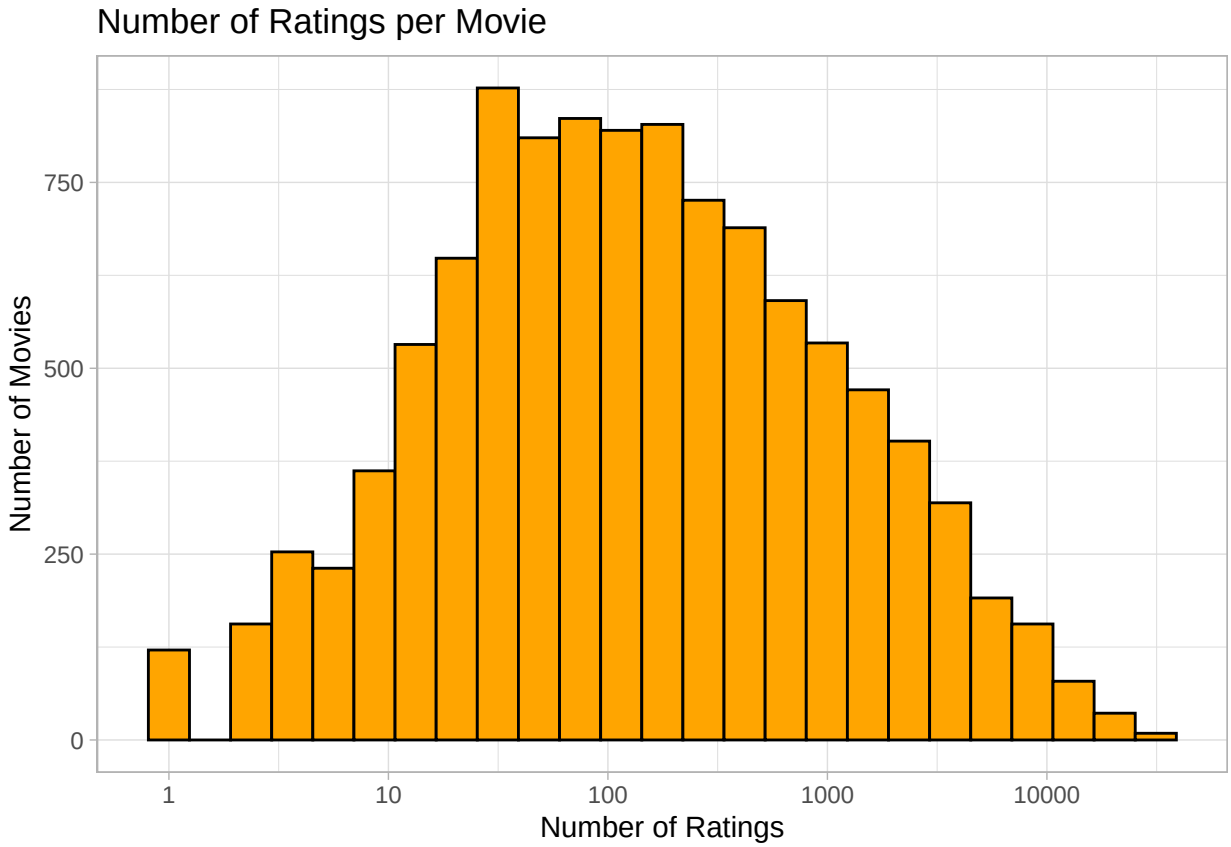
Before we even look at the distribution for movie ratings and its users, natural human behavior would give us insight to the fact that users have preferences in which movies to watch and which to rate. But does this mean we will see an even distribution of ratings? Psychology would suggest no - but does the data agree? The ratings distribution chart below provides support in favor of agreement.

```
edx %>% ggplot(aes(rating)) + geom_histogram(binwidth = .25, fill = "orange", color = "black") +
  scale_x_discrete(limits = c(seq(.5,5,.5))) +
  scale_y_continuous(breaks = c(seq(0, 3000000, 500000))) +
  ggtitle("Ratings Distribution") + xlab("Rating") +
  ylab("Number of Movies Rated") + theme_light()
```



Movies are also not reviewed equally. Some have been reviewed far more often than others while some have very few or even one rating. This is significant to note as it may influence our model. Let's see how many movies have only been rated once!

```
edx %>% count(movieId) %>% ggplot(aes(n)) +
  geom_histogram(bins = 25, fill = "orange", color = "black") +
  scale_x_log10() + xlab("Number of Ratings") + ylab("Number of Movies") +
  ggtitle("Number of Ratings per Movie") + theme_light()
```



Wow, 121 movies have been rated only once! This is important to note. In order to increase accuracy and lessen the chance of returning unreliable predictions, regularization and a penalty term will be applied in later models.

Regularization is a technique used to reduce error by fitting a function appropriately on the training dataset and avoid overfitting. This will help tune the calculation by adding a penalty term in the error function. This also aids in controlling excessive fluctuations in the function to limit extreme values.

Now, let's view some movies that were only rated once. What reasons could they possibly have to receive only one rating? Initially, it seems awfully unlikely to only receive one rating for a movie.

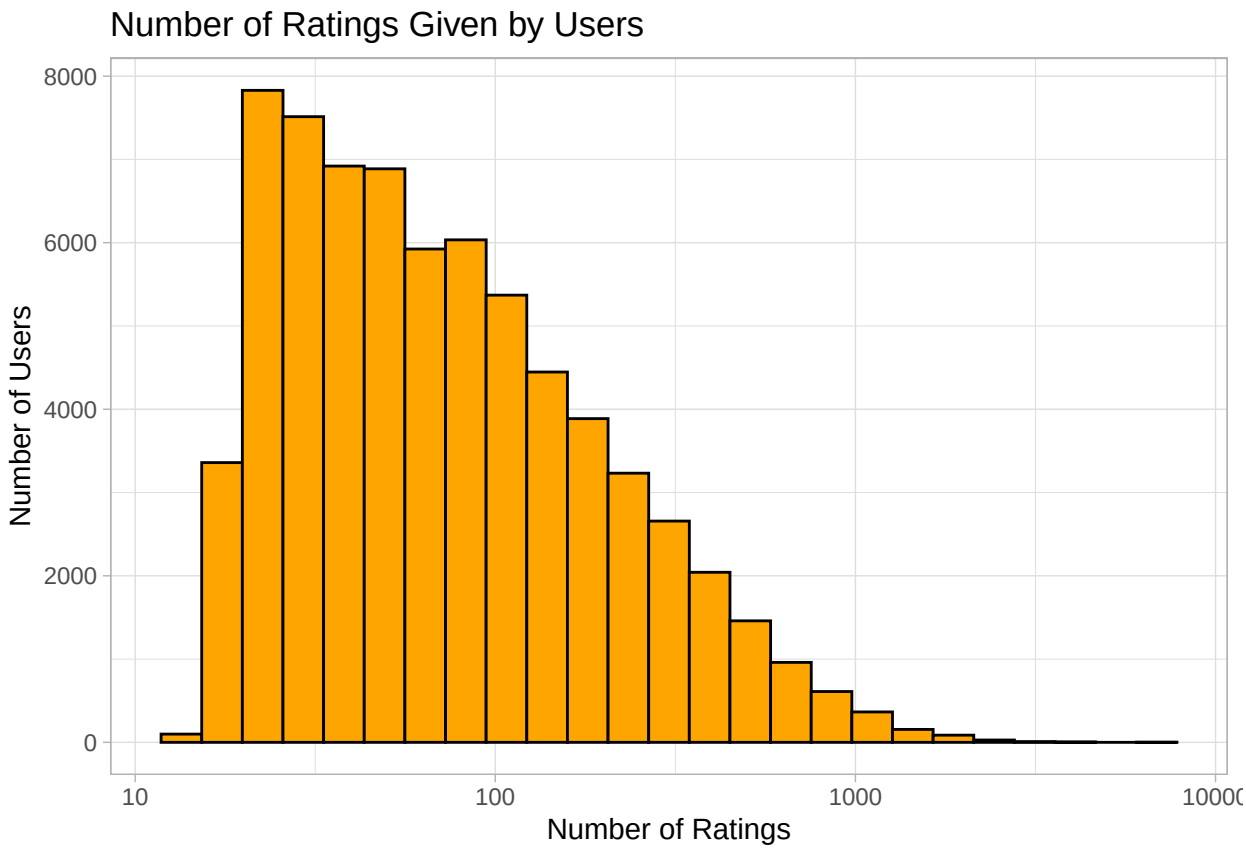
```
edx %>% group_by(movieId) %>% summarize(count = n()) %>% filter(count == 1) %>%
  left_join(edx, by = "movieId") %>% group_by(title) %>%
  summarize(rating = rating, n_rating = count) %>% slice(1:10) %>% kable()
```

title	rating	n_rating
100 Feet (2008)	2.0	1
4 (2005)	2.5	1
5 Centimeters per Second (Byôsoku 5 senchimêtoru) (2007)	3.5	1
Accused (Anklaget) (2005)	0.5	1
Ace of Hearts (2008)	2.0	1
Ace of Hearts, The (1921)	3.5	1
Adios, Sabata (Indio Black, sai che ti dico: Sei un gran figlio di...) (1971)	1.5	1
Africa addio (1966)	3.0	1
Archangel (1990)	2.5	1
Bad Blood (Mauvais sang) (1986)	4.5	1

After reviewing this table, we can see that these movies appear to be esoteric and little-known to the general public. Having only one rating will practically make it impossible to predict future ratings for them. Even movies with a small number of ratings will be difficult to predict.

Our next objective is to look at the number of ratings given by users. This will aid in model development and provide insight on to methods like regularization.

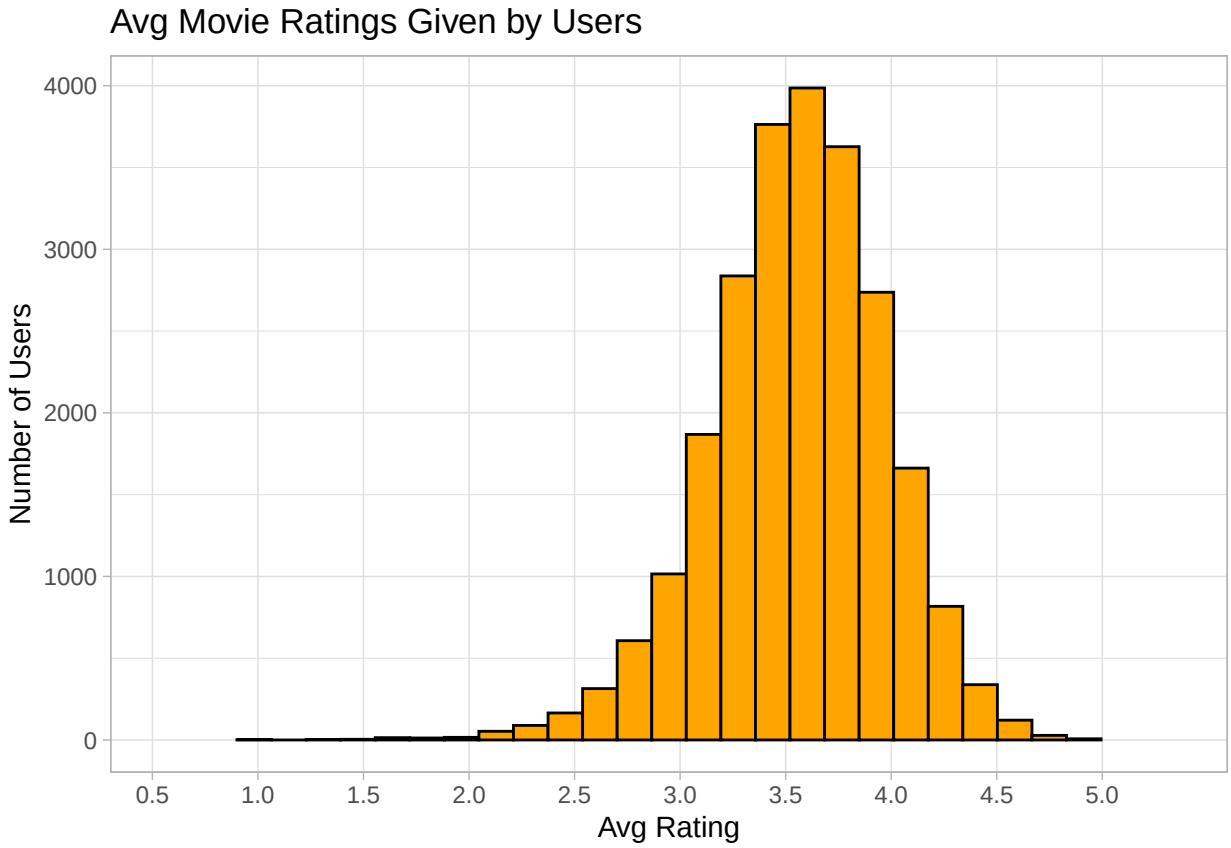
```
edx %>% count(userId) %>% ggplot(aes(n)) +
  geom_histogram(bins = 25, fill = "orange", color = "black") +
  scale_x_log10() + xlab("Number of Ratings") + ylab("Number of Users") +
  ggtitle("Number of Ratings Given by Users") + theme_light()
```



This chart demonstrates visually that a majority of users have rated between 30 and 100 movies - necessitating the introduction of a penalty term into our future models.

Now we should look at how these users have performed in their ratings (i.e., How critical are they? How much variation is there?)

```
edx %>% group_by(userId) %>% filter(n() >= 100) %>%
  summarize(beta_u = mean(rating)) %>% ggplot(aes(beta_u)) +
  geom_histogram(bins = 25, fill = "orange", color = "black") +
  xlab("Avg Rating") + ylab("Number of Users") +
  ggtitle("Avg Movie Ratings Given by Users") +
  scale_x_discrete(limits = c(seq(.5,5,.5))) + theme_light()
```



This visualization shows that users have varying levels of how critical they are towards their movie ratings. Some users have an affinity for lower ratings while the opposite of is true for another set of users. The chart was limited to those users that have rated at least 100 movies.

Constructing the Models

The RMSE is how machine learning algorithms measure models for accuracy. The function to be applied is as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{u,i}(\hat{y}_{u,i} - y_{u,i})^2}$$

The RMSE is a typical error that is made when predicting, such as movie ratings. An error > 1 would be a bad predictor of movie ratings. This means the prediction would be wrong by more than 1 star! Thus, the lower the error, the better prediction.

Average Movie Rating Model

This first model will be a simple prediction for the dataset. Calculating the mean is our first step - it is expected to reside between 3 and 4 based on what little we currently know. The simple prediction will purely be recommending the same rating for all movies regardless of the user giving it. This would imply that all variations in movie ratings are explained solely by random variation, and nothing else.

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

This is where μ would be the true rating and $\epsilon_{u,i}$ is the random variation.

```r
mu <- mean(edx$rating)
mu %>% kable()
```

x
3.512464

Then, we shall predict all the unknown ratings with μ or mu to obtain our simple RMSE:

```r
simple_rmse <- RMSE(validation$rating, mu)
rmse_preds <- data.frame(method = "Simple Movie Rating Model", RMSE = simple_rmse)
rmse_preds %>% kable()
```

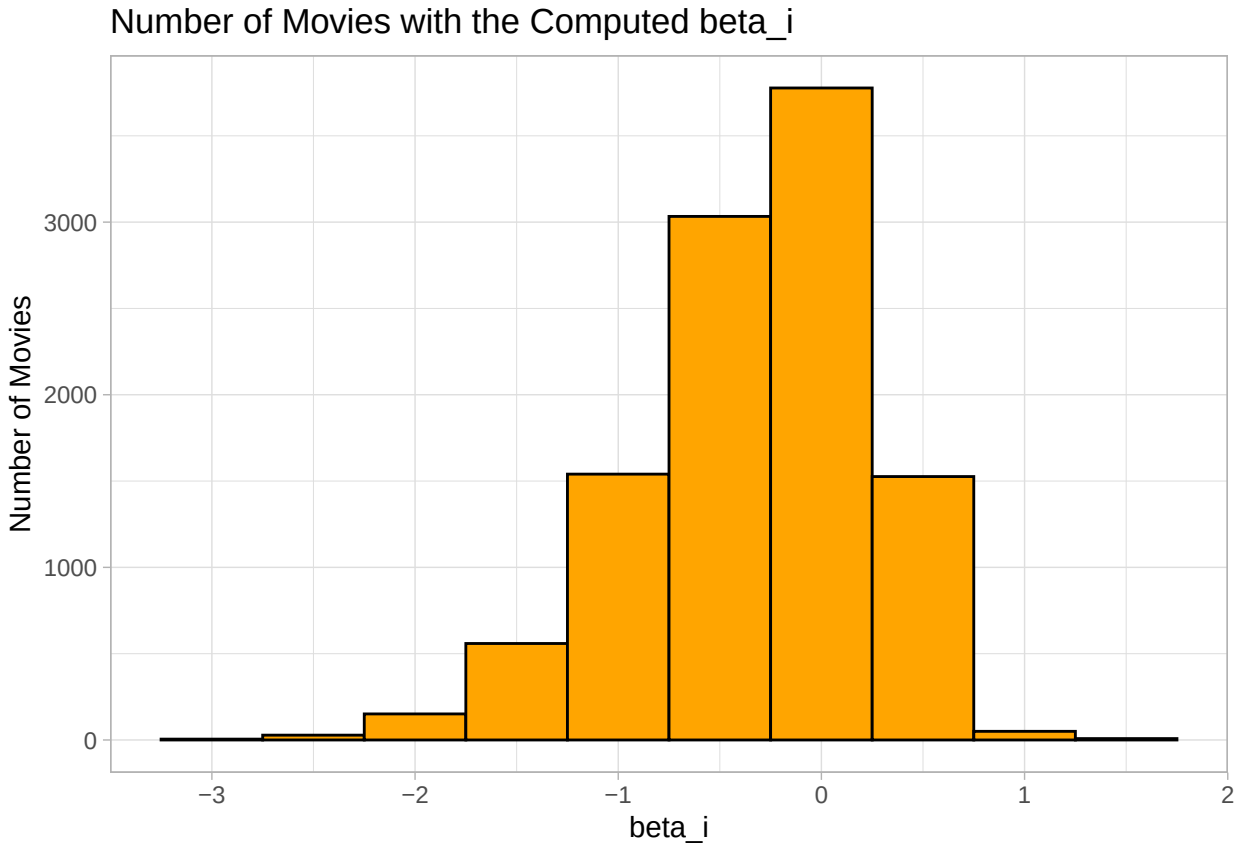
method	RMSE
Simple Movie Rating Model	1.060651

This provides us with our baseline in developing a good machine learning algorithm to predict movie ratings. Notice that it is above 1; this is not good. Here, we apply what we learned from our earlier exploratory data analysis.

Type of Movie Effect Model

From general human experience, we know that some movies will generally score higher ratings than others. This is usually linked to the popularity of movies amongst users. Therefore, we must calculate the disparity of each movies' average (mean) rating and μ that we determined above. This will leave us with the bias effect for each movie stored in the beta_i variable below.

```r
movie_avgs <- edx %>% group_by(movieId) %>% summarize(beta_i = mean(rating - mu))
movie_avgs %>% ggplot(aes(beta_i)) + geom_histogram(bins = 10, fill = "orange", color = "black") +
  xlab("beta_i") + ylab("Number of Movies") + ggtitle("Number of Movies with the Computed beta_i") +
  theme_light()
```



It is easy to see that the resulting histogram is not perfectly centered. It shows that more movies have lesser ratings than the μ; thus, beta_i has negative effects.

Thus, if we introduce the penalty term, our prediction should improve.

```
penalty_ratings <- mu + validation %>%  left_join(movie_avgs, by = "movieId") %>% pull(beta_i)
first_rmse <- RMSE(penalty_ratings, validation$rating)
rmse_preds <- add_row(rmse_preds, method = "Movie Effect Model", RMSE = first_rmse)
rmse_preds %>% kable()
```

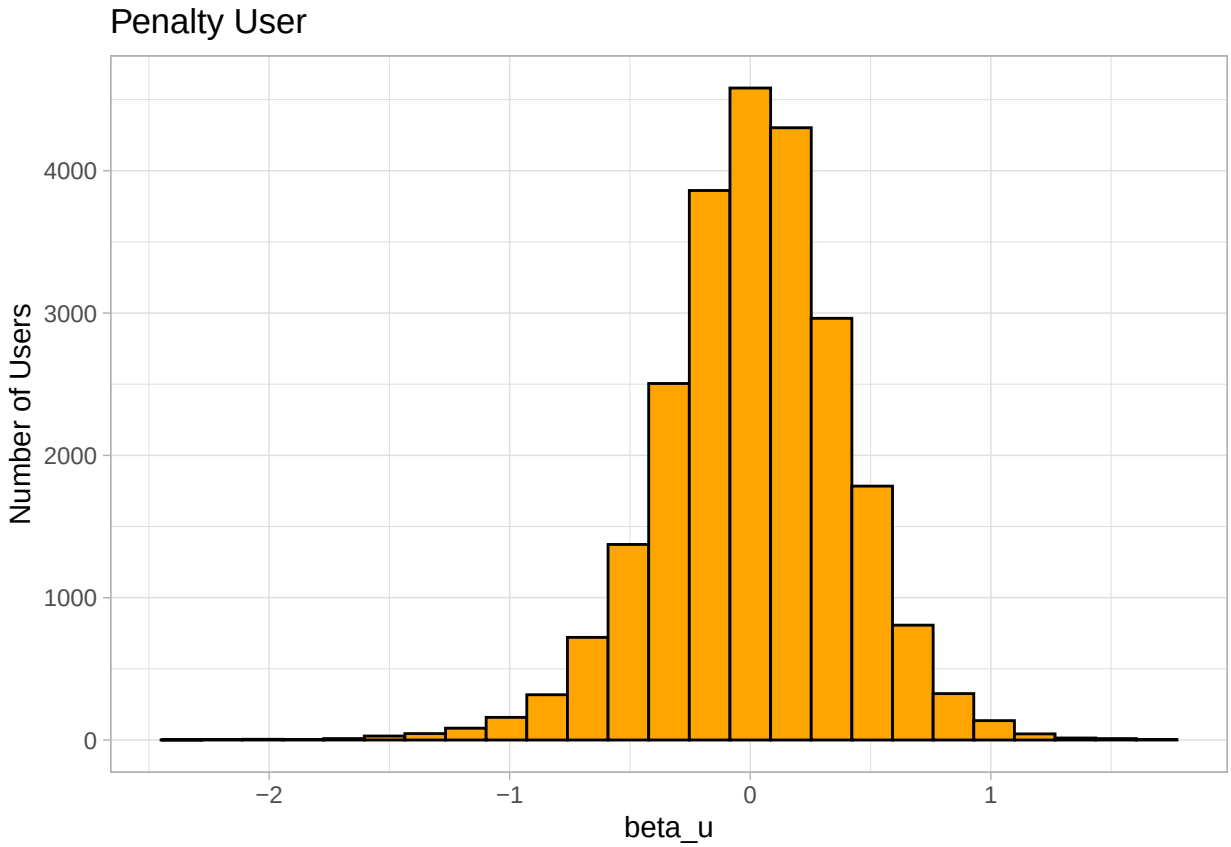
method	RMSE
Simple Movie Rating Model	1.0606506
Movie Effect Model	0.9437046

Here we were able to predict movie ratings based on the fact that movies are rated differently from each other. By adding beta_i to μ, we can account for this difference in the specific movie average from the overall average of movies. Were we able to improve our algorithm? Yes! We successfully brought it under an RMSE error of 1, but it is still not very accurate as there are other variables to consider.

Add User Effect to Movie Effect Model

Now, we can add the variable of the User Effect - similar to the Movie Effect - to our model. This will look at individual users that have rated at least 100 movies to create the penalty term user effect.

```
user_avgs <- edx %>% left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>% filter(n() >= 100) %>% summarize(beta_u = mean(rating - mu - beta_i))
user_avgs %>% ggplot(aes(beta_u)) +
  geom_histogram(bins = 25, fill = "orange", color = "black") + xlab("beta_u") +
  ylab("Number of Users") + ggtitle("Penalty User") + theme_light()
```



As expected, there is wide variability across the user spectrum. Therefore, if we add the user effect (user penalty term) to our equation, this may show further improvement to our machine learning algorithm. We can do this by shifting our equation from the Movie Effect Model:

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

To the Movie and User Effect Model:

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

Similar to the Movie Effect penalty term, if beta_u is negative, this would show that a highly rated movie was given a poor rating from this user.

We can then compute for beta_u:

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

```
user_avgs <- edx %>% left_join(movie_avgs, by='movieId') %>% group_by(userId) %>%
  summarize(beta_u = mean(rating - mu - beta_i))
```

Let's calculate to see if the RMSE improves:

```
predicted_ratings <- validation %>% left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>% mutate(prediction = mu + beta_u + beta_i) %>%
  pull(prediction)
second_rmse <- RMSE(predicted_ratings, validation$rating)
rmse_preds <- add_row(rmse_preds, method = "Movie and User Effect Model", RMSE = second_rmse)
rmse_preds %>% kable()
```

method	RMSE
Simple Movie Rating Model	1.0606506
Movie Effect Model	0.9437046
Movie and User Effect Model	0.8655329

Our RMSE rating prediction algorithm improved! However, there may still be outliers on the extremities of the movie ratings spectrum. These - combined with obscure movies or movies with fewer users giving ratings - can increase the errors in our RMSE calculation. To decrease the likelihood of errors we penalize the outliers to minimize their impact. We can also add regularization to provide insurance against overfitting. In statistics, overfitting is defined as the production of an analysis that corresponds too closely - or exactly - to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably.

Regularized Movie and User Effect Model

The penalty terms are estimates created by movies with very few ratings and users with very few given ratings. Why are these important? It is because they can have disproportionately strong influence over the overall prediction of the machine learning algorithm. As described, regularization will aid in minimizing their influence. The tuning parameter is known as lambda. It is used by repeating the process of slightly larger lambda values to then evaluate how it affects the variability of the RMSE prediction.

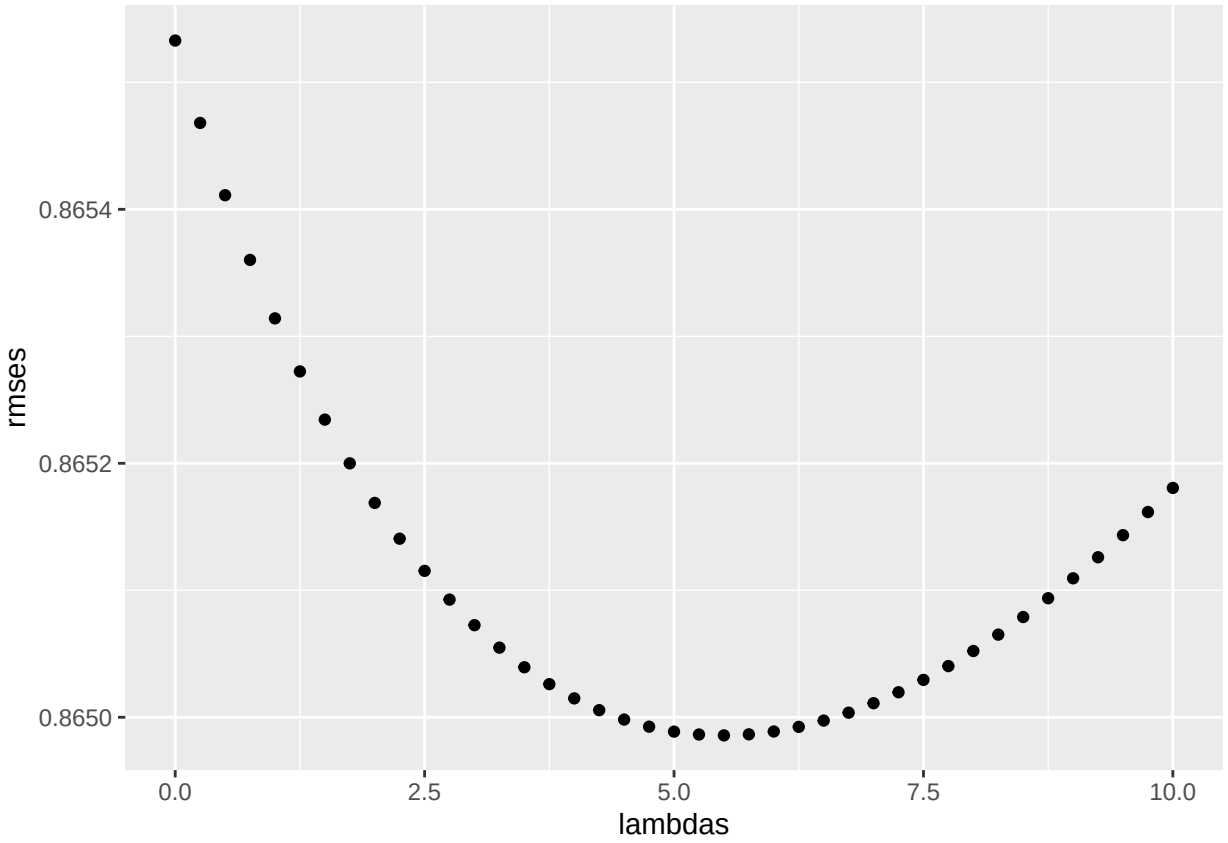
WARNING: If you decide to run this code in your RStudio, or other software, this may be a very slow calculation - please be patient.

```
lambdas <- seq(0,10,.25)

rmses <- sapply(lambdas, function(lambda){
  mu <- mean(edx$rating)
  beta_i <- edx %>% group_by(movieId) %>% summarize(beta_i = sum(rating - mu) / (n() + lambda))
  beta_u <- edx %>% left_join(beta_i, by="movieId") %>% group_by(userId) %>%
    summarize(beta_u = sum(rating - beta_i - mu) / (n() + lambda))
  predicted_ratings <- validation %>% left_join(beta_i, by = "movieId") %>%
    left_join(beta_u, by = "userId") %>%
    mutate(pred = mu + beta_i + beta_u) %>% pull(pred)
  lambda_predict <- RMSE(predicted_ratings, validation$rating)
  lambda_predict
})
```

To find the optimal lambda, we plot RMSE against the lambdas:

```
ggplot(mapping = aes(x = lambdas, y = rmses)) + geom_point()
```



To mathematically discover the optimal lambda to support the plot, we use:

```
op_lambda <- lambdas[which.min(rmses)]
op_lambda %>% kable()
```

x
5.5

Thus, the complete model will incorporate the optimal lambda: 5.5

Let's check out our RMSE accuracy now:

```
rmse_preds <- add_row(rmse_preds, method =
    "Regularized Movie and User Effect Model", RMSE = min(rmses))
rmse_preds %>% kable()
```

method	RMSE
Simple Movie Rating Model	1.0606506
Movie Effect Model	0.9437046
Movie and User Effect Model	0.8655329
Regularized Movie and User Effect Model	0.8649857

Results

The end results for our RMSE values are:

```
rmse_preds %>% kable()
```

method	RMSE
Simple Movie Rating Model	1.0606506
Movie Effect Model	0.9437046
Movie and User Effect Model	0.8655329
Regularized Movie and User Effect Model	0.8649857

Conclusion

We were able to get the RMSE down to 0.8649857! This shows that the model works pretty well as a machine learning algorithm to predict movie ratings based on the MovieLens Dataset. The optimal model thus far is based on the Regularized Movie and User Effect Model as it has the lowest RMSE value. This was successful in reaching our project's goal of a model producing an RMSE lower than 0.865. We may be able to improve upon this algorithm through evaluating other variables, regularization methods, or others. Hardware limitations were difficult to manage, such as RAM and CPU, but successful nonetheless.