MovieLens

Austin Murr

12/23/2020

MovieLens Data Science Capstone

Overview

Project Goal

The goal for this project is to be able to predict what movies a user will enjoy. The criterion against which we will be testing our model is the Root Mean Square Error (RMSE), which is defined as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n} e_t^2}$$

The target value for using our model on our test set is *RMSE < 0.86490*.

Dataset

We will be using the 10 Million Entries version of the MovieLens dataset. This set will be divided so that 90% of the data is used for training, and 10% will be used for testing our model.

The data is structured as follows:

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	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
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

Here are some important details derived from this quick look:

1. Each user can rate multiple movies. (Many-to-Many relationship)
2. Each movie can be rated by multiple users. (Many-to-Many relationship)
3. The rating is given on a scale from 0 to 5 with 0.5 step increments.
4. The date and time of the rating is recorded in the Unix Epoch time format.
5. The title of each movie includes the year in which it was released in a set of parenthesis at the very end of the content.
6. Each movie can have zero, one, or multiple genres. They are all included in a single character string separated by a pipe.

The first and second observations can be summarized as follows:

Users	Movies
69878	10677

We can also compile a list of all the possible genres listed:

```
##  [1] "(no genres listed)" "Action"             "Adventure"
##  [4] "Animation"          "Children"           "Comedy"
##  [7] "Crime"              "Documentary"        "Drama"
## [10] "Fantasy"            "Film-Noir"          "Horror"
## [13] "IMAX"               "Musical"            "Mystery"
## [16] "Romance"            "Sci-Fi"             "Thriller"
## [19] "War"                "Western"
```

Key Steps

The approach taken was one of adding one feature at a time to improve the algorithm. Here are some key steps to look for on the way:

1. Data Cleaning
2. Exploratory Data Analysis
3. Naive Mean Model
4. Mean + Movie Bias Model
5. Mean + Movie Bias + User Bias Model
6. Regularized Mean + Movie Bias + User Bias Model
7. Mean + Movie Bias + User Bias + Release Year Bias Model
8. Regularized Mean + Movie Bias + User Bias + Release Year Bias + Number of Genres Bias Model
9. Lambda Optimization
10. Results
11. Conclusion

Methods

Data Cleaning

In our first look at the data, we noticed a couple of features that could be improved. Here is our agenda for data cleaning:

- Extract release year from the movie title
- Convert the `genre` column from `character` class to a `list` class
- Transform timestamp into a human-readable format

It is not yet clear which of these features will actually be of value, but cleaning the data first gives us the opportunity to explore which data matters most.

As an important note, we need to apply the same cleaning process to both the training and testing sets to maintain consistency and interoperability.

Make Timestamp of Rating Human Readable First we convert it to the YYYY-MM-DD format:

```
edx$timestamp = as_datetime(edx$timestamp) %>% date()
validation$timestamp = as_datetime(validation$timestamp) %>% date()
head(edx$timestamp)
```

```
## [1] "1996-08-02" "1996-08-02" "1996-08-02" "1996-08-02" "1996-08-02"
## [6] "1996-08-02"
```

We will now parse these into separate year, month, and day columns:

```
edx$rateYear = edx$timestamp %>% year()
edx$rateMonth = edx$timestamp %>% month()
edx$rateDay = edx$timestamp %>% day()

edx = edx %>% select(-timestamp)

validation$rateYear = validation$timestamp %>% year()
validation$rateMonth = validation$timestamp %>% month()
validation$rateDay = validation$timestamp %>% day()

validation = validation %>% select(-timestamp)
```

Extract Release Year Using regex, we can extract the year in which each movie was released. This will become its own column, and we will then remove the year from the title in order to make the data more clear and tidy.

```
edx$release = edx$title %>% str_match("\\s\\(\\d{4}\\)") %>%
  str_remove("\\s") %>% str_remove("\\(") %>% str_remove("\\)") %>% as.numeric()
edx$title = edx$title %>% str_remove("\\s\\(\\d{4}\\)")

validation$release = validation$title %>% str_match("\\s\\(\\d{4}\\)") %>%
  str_remove("\\s") %>% str_remove("\\(") %>% str_remove("\\)") %>% as.numeric()
validation$title = validation$title %>% str_remove("\\s\\(\\d{4}\\)")
```

Convert Genre Column to List The final and most computationally expensive part of cleaning our data is making it easier to process the genre data. Since each movie can have zero, one, or multiple genres, it would make more sense if the genre data class was a list instead of a character string:

```
edx$genres = strsplit(edx$genres, "\\|")
validation$genres = strsplit(validation$genres, "\\|")
```

The issue with this format is we still have to iterate through each item of each entry's item, which is rather inefficient. To solve this, we will create a column for each genre, for which each movie will either record a `TRUE` or `FALSE`.

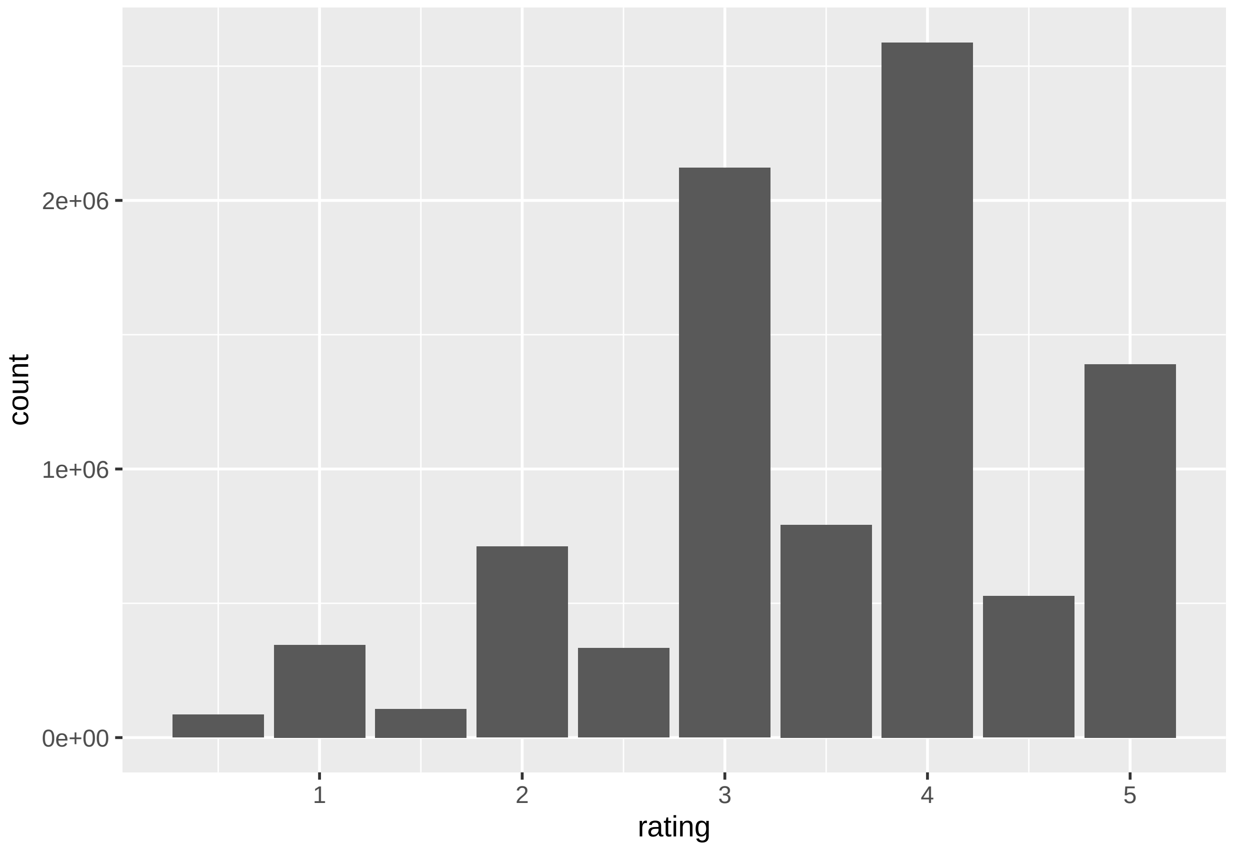
```
for(value in genre_list)  {
  gen_vector = vector(length = NROW(edx))
  for(i in seq(NROW(edx)))  {
    ifelse(value %in% edx$genre[[i]], {gen_vector[i] = TRUE}, {gen_vector[i] = FALSE})
  }
  edx[[value]] = gen_vector
}

for(value in genre_list)  {
  gen_vector = vector(length = NROW(validation))
  for(i in seq(NROW(validation)))  {
    ifelse(value %in% validation$genre[[i]], {gen_vector[i] = TRUE}, {gen_vector[i] = FALSE})
  }
  validation[[value]] = gen_vector
}

edx = edx %>% select(-genres)
validation = validation %>% select(-genres)

num_edx = edx[,9:28] %>% rowSums()
edx = edx %>% mutate(NUM = num_edx)

num_valid = validation[,9:28] %>% rowSums()
validation = validation %>% mutate(NUM = num_valid)

write_delim(edx, "edx.csv", delim = "\\")
write_delim(validation, "validation.csv", delim = "\\")

edx[,1:9] %>% head() %>% kable()
```

userId	movieId	rating	title	rateYear	rateMonth	rateDay	release	Comedy
1	122	5	Boomerang	1996	8	2	1992	TRUE
1	185	5	Net, The	1996	8	2	1995	FALSE
1	292	5	Outbreak	1996	8	2	1995	FALSE
1	316	5	Stargate	1996	8	2	1994	FALSE
1	329	5	Star Trek: Generations	1996	8	2	1994	FALSE
1	355	5	Flintstones, The	1996	8	2	1994	TRUE

Exploratory Data Analysis

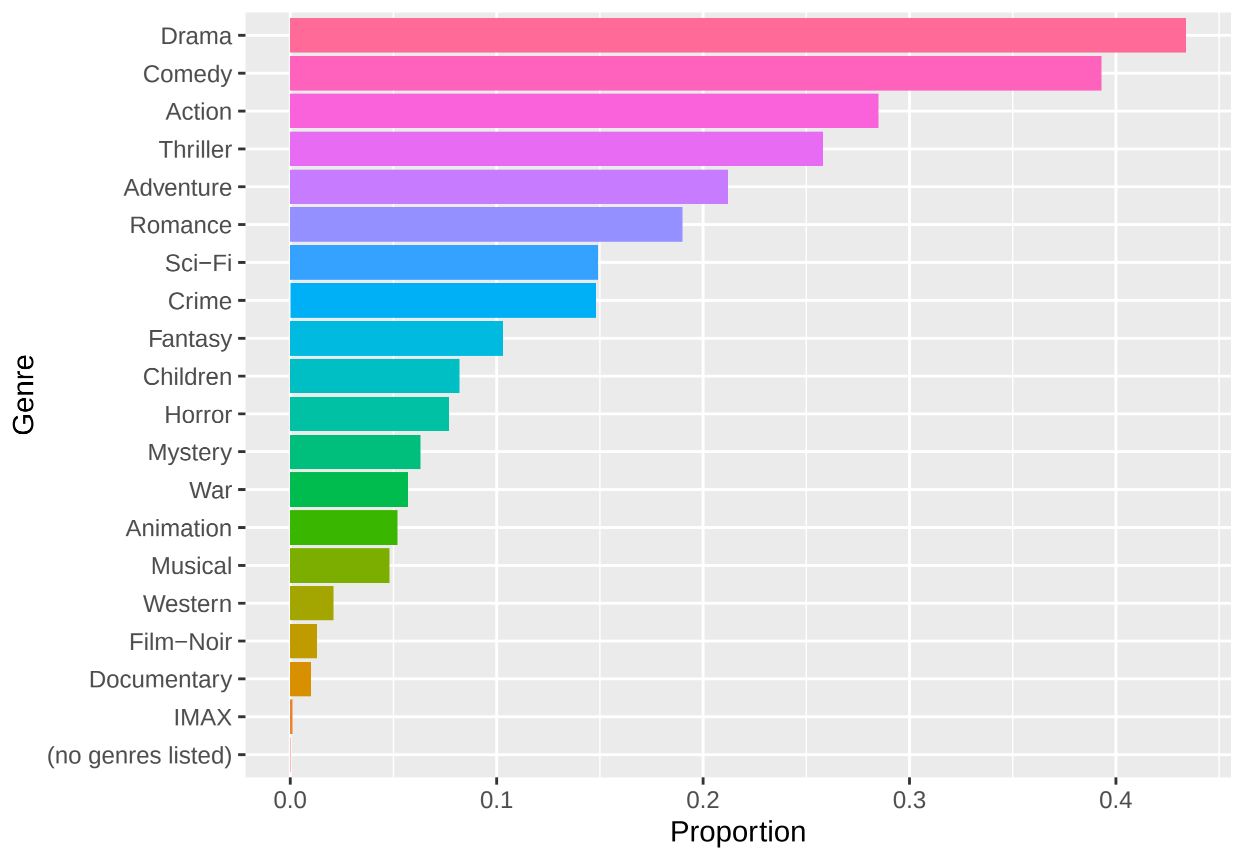
The next order of business is getting a grasp of trends in the data. We do this by using statistical methods and visualizing the data.

Let's take a look at the distribution of ratings:

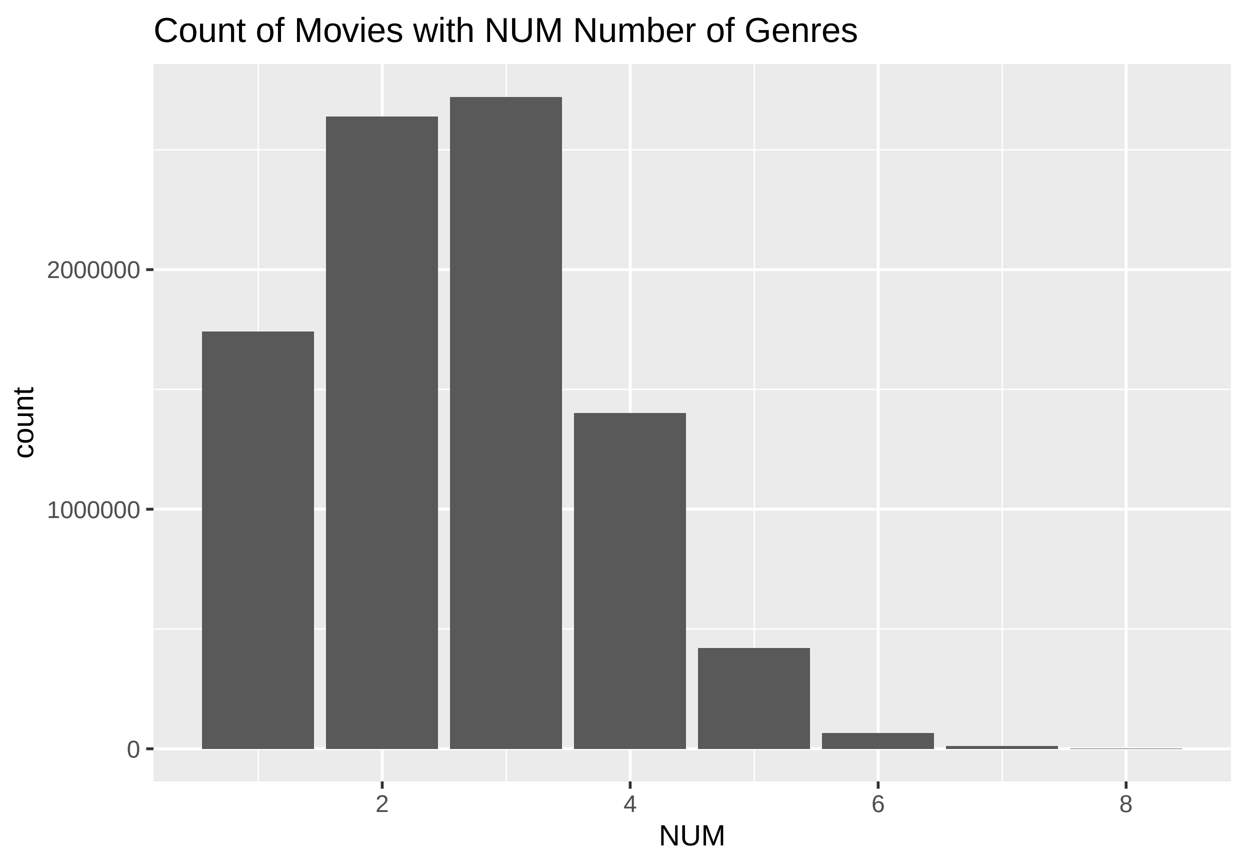


Whole number ratings are vastly more common than half-number ratings. Though this makes it a bit harder to visualize, we can still see that the most common ratings are in the 3 - 4 point range.

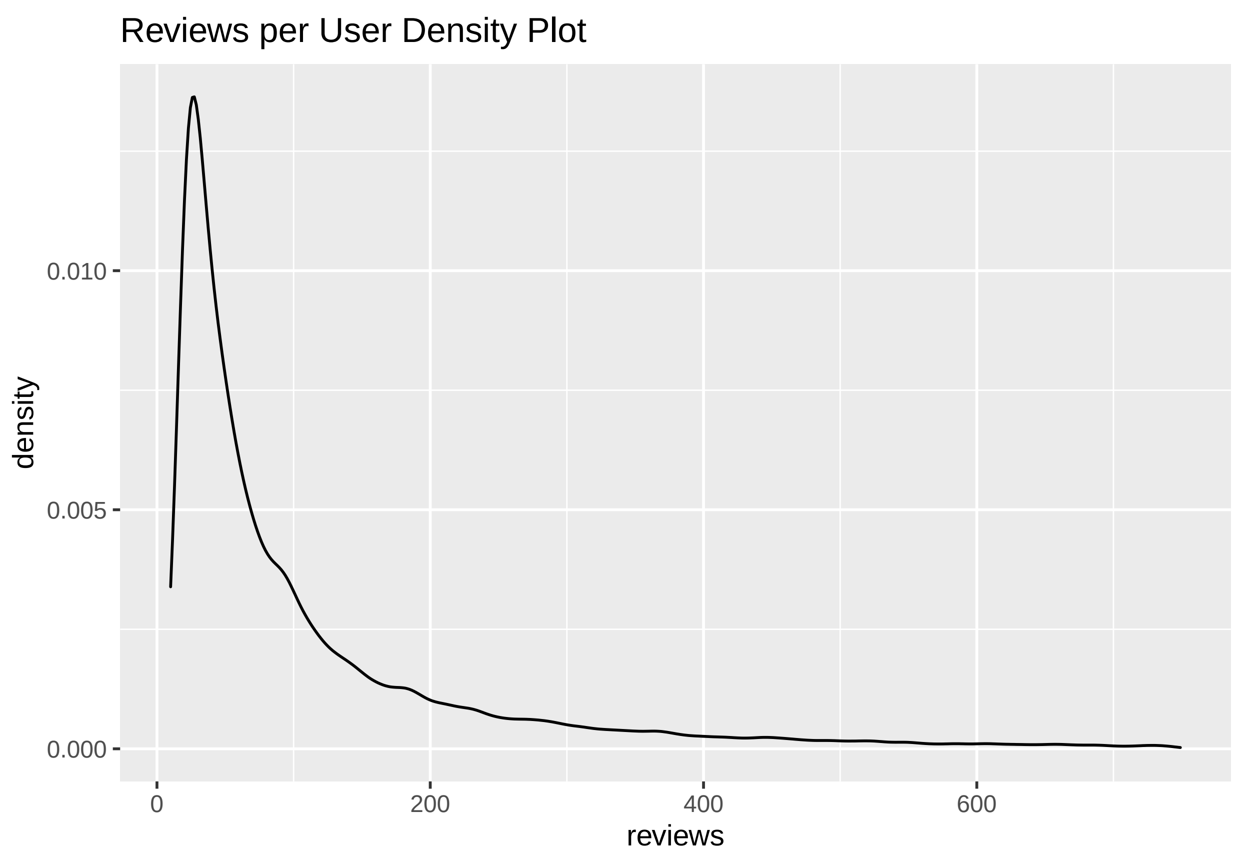
One of the most obvious insights is determining which genres are most prevalent. In the following code and graph, the `Proportion` refers to the proportion of movies that list the genre. Remember, movies can have multiple genres, so the sum of total proportions will be greater than 1.00.



Almost 45% of movies fall within the `Drama` genre, and nearly 40% are labeled as a `Comedy`. Movies can also have multiple genres, with the breakdown presented below.



Likewise, the number of movies each user rates varies.



To summarize the insight gained:

1. The most common ratings are in the 3 - 4 range
2. Movies can have multiple genres
3. Users can rate multiple movies

While this may not seem like much, we will expand upon this in the modeling section.

Modeling

Navie Mean Model

For the first model, we will simply guess the mean rating for all movies.

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

As a reminder, the goal is to minimize the RMSE equation, given by this code:

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

We will first define μ.

```
mu_hat = mean(edx$rating)
mu_hat
```

```
## [1] 3.512465
```

```
naive_RMSE = RMSE(edx$rating, mu_hat)
naive_RMSE
```

```
## [1] 1.060331
```

Mean + Movie Bias Model

The good news is that we have multiple variables that affect the rating, one of which is the movie itself.

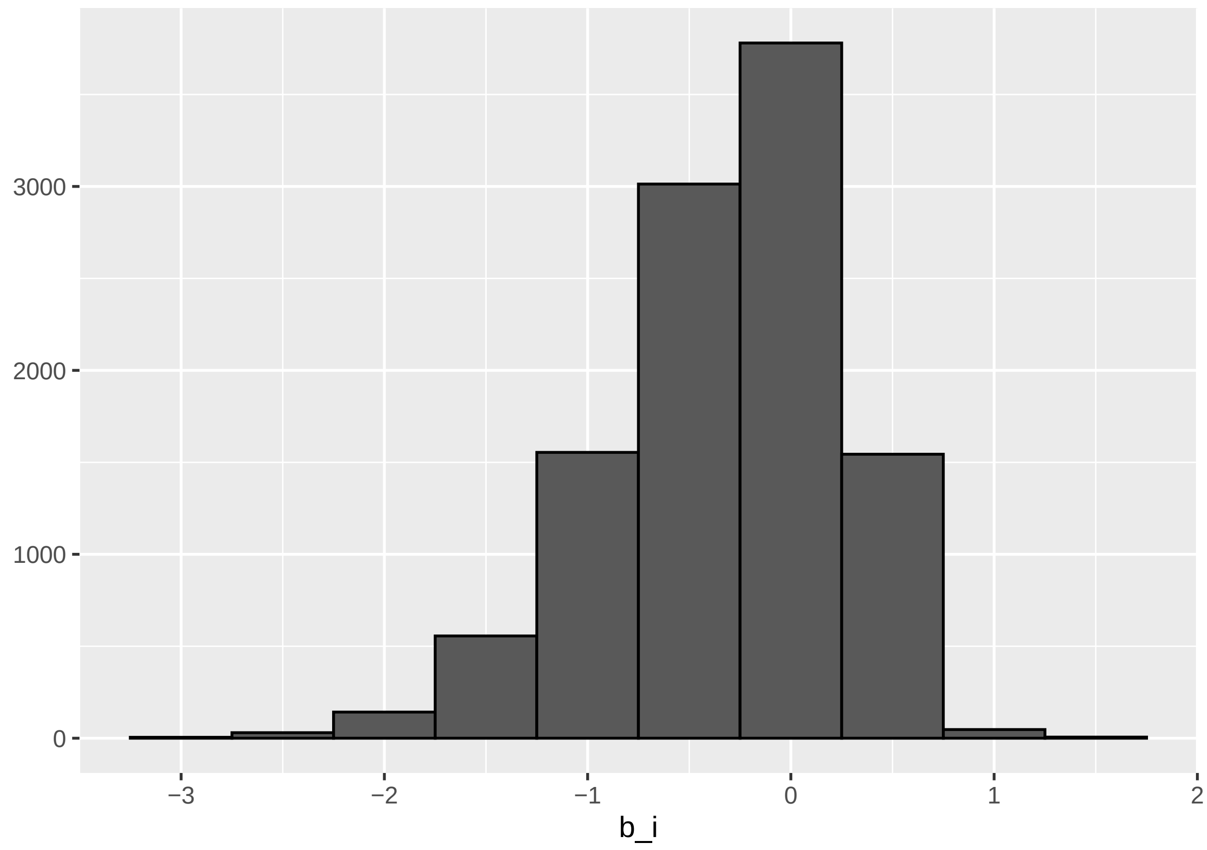
To incorporate this movie bias, we will introduce another term to our equation:

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

Now, we can think of the movie bias as being how much better or worse it's own average rating is compared to the average for all movies. In other words:

$$b_i = \mu_i - \mu$$

Here is a chart that details the count of each movies' variance from μ.



Combining this movie bias into our model improves it significantly:

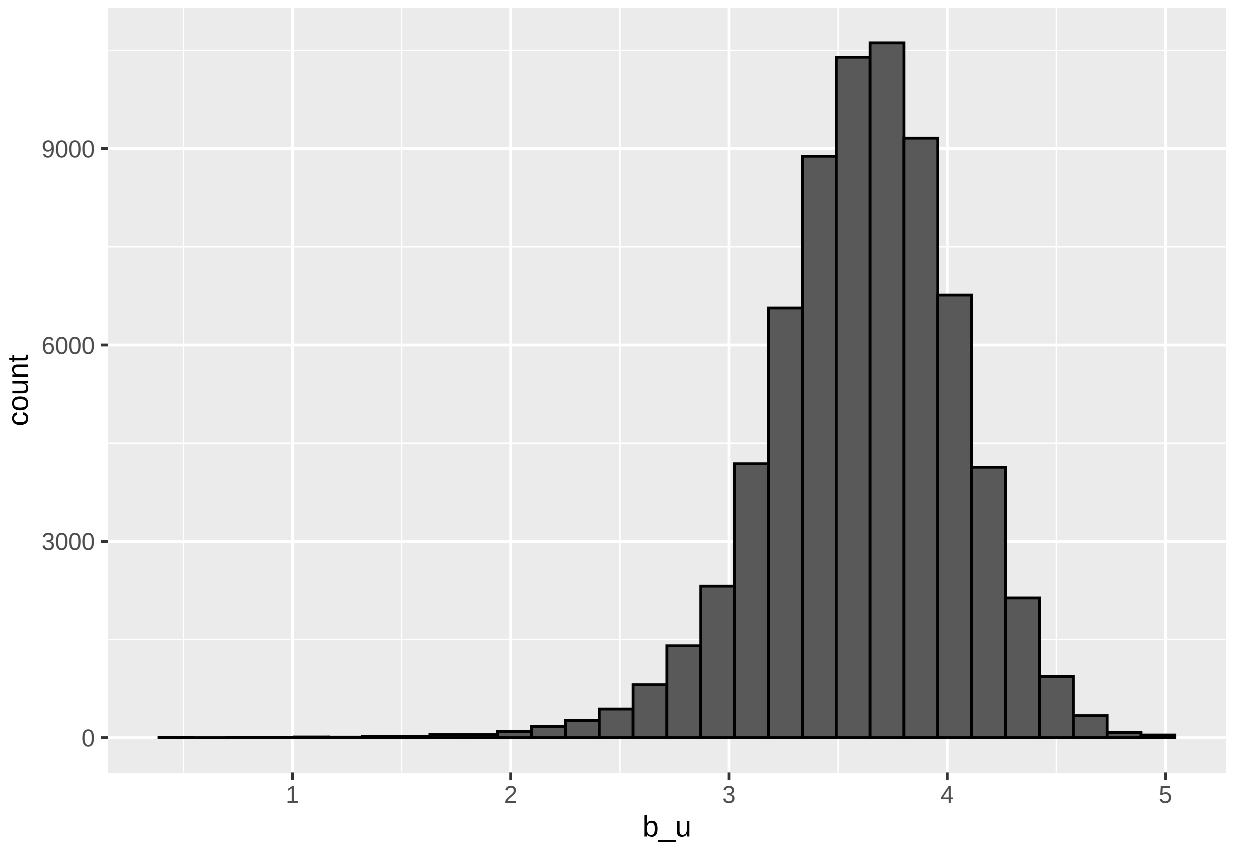
```r
predicted_ratings = mu_hat + edx %>% left_join(movie_avgs, by='movieId') %>% pull(b_i)
RMSE(predicted_ratings, edx$rating)
```

```
## [1] 0.9423475
```

Mean + Movie Bias + User Bias Model

In addition to bias among movies, users also tend to have different rating habits.

```r
edx %>%
group_by(userId) %>%
summarize(b_u = mean(rating)) %>%
filter(n()>=100) %>%
ggplot(aes(b_u)) +
geom_histogram(bins = 30, color = "black")
```



As we can see, most people typically give out average ratings, but other people are very easy to impress and others are very hard to please. This User Bias can be incorporated into the model to further improve our RMSE scores.

To incorporate user bias, our equation will look like this:

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

```
user_avgs = edx %>% left_join(movie_avgs, by='movieId') %>% group_by(userId) %>% summarize(b_u = mean(ra
```

The impact of this additional term is hard to overstate. It dropped our RMSE score by more than 10%!

```
## [1] 0.8567039
```

Now, this RMSE score is acceptable and even meets our goal, but let's see if we can do better.

When we take a look at our biggest mistakes and the number of ratings they recieved, we realize that we are giving undeserved weight to just a few users:

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


edx %>% count(movieId) %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  slice(1:10) %>%
  pull(n) %>% kable()
```

x
1
2
1
1
1
1
4
4
4
2

The impact of this is that obscure movies are rated as the best due to the unequal weight these few users control. Here is a list of what our model describes as the 10 best movies:

```
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  slice(1:10) %>%
  pull(title) %>% kable()
```

x
Hellhounds on My Trail
Satan's Tango (Sátántangó)
Shadows of Forgotten Ancestors
Fighting Elegy (Kenka erejii)
Sun Alley (Sonnenallee)

x
Blue Light, The (Das Blaue Licht)
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva)
Human Condition II, The (Ningen no joken II)
Human Condition III, The (Ningen no joken III)
Constantine's Sword

These do not make any sense.

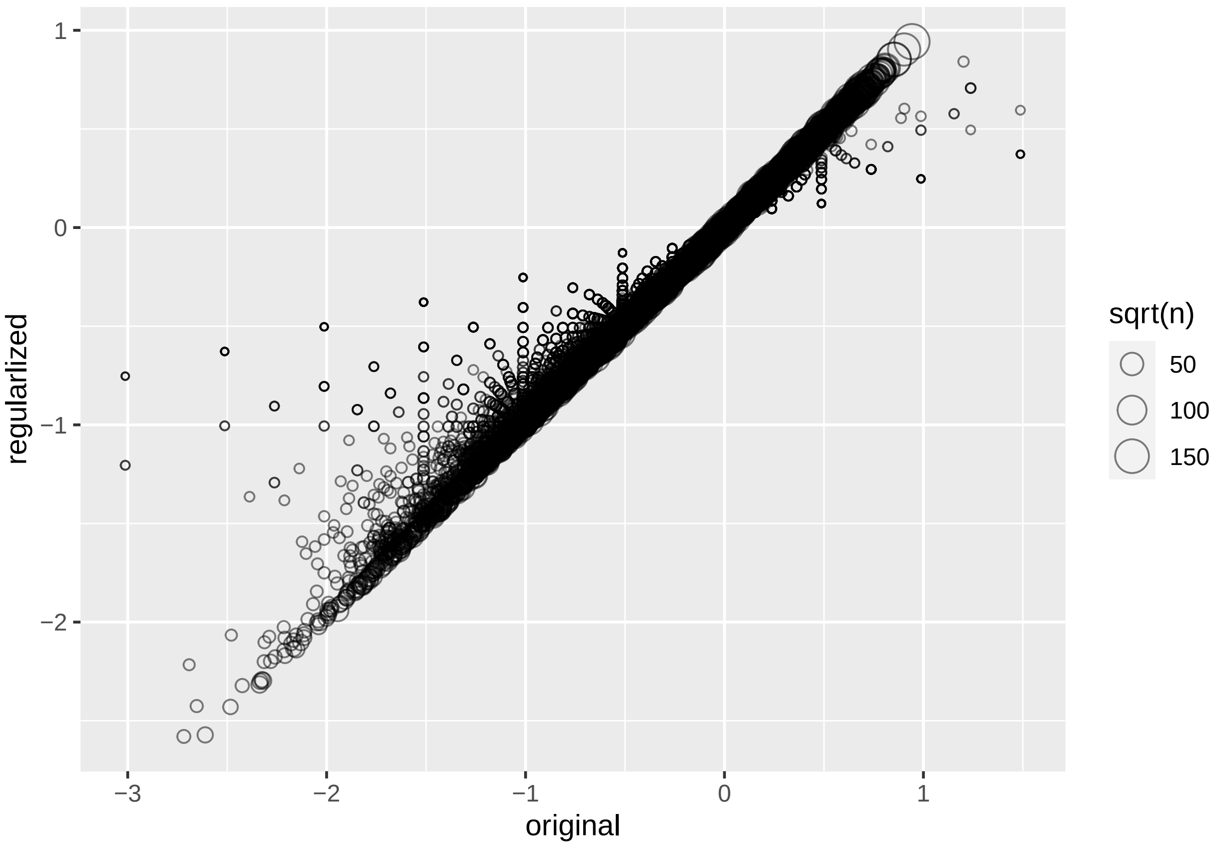
Regularized Mean + Movie Bias + User Bias Model

One way to correct this is through regularization. Essentially, we add a penalty term for movies that do not have many observations. Our penalty term will be defined as follows:

$$\sum_{n}^{i} \frac{b_i - \mu}{n + \lambda}$$

Essentially, lambda affects the number of ratings proportional to the number of ratings. Large numbers are barely affects while smaller ones are greatly affected. By putting this in the denominator, we assign its impact to the residuals of the rating.

Here is a comparison between the original and regularized residuals.



As this shows, regularization has helped distribute influence more equitably.

Now, our top ten movies are:

x
Shawshank Redemption, The
Godfather, The
Usual Suspects, The
Schindler's List
More
Casablanca
Rear Window
Sunset Blvd. (a.k.a. Sunset Boulevard)
Third Man, The
Double Indemnity

This makes sense, because movies like The Shawshank Redemption, Casablanca, The Godfather, and Schindler's List are all often considered the best movies of all time.

Mean + Movie Bias + User Bias + Release Year Bias Model

Now, adding more terms can help us out even more. This time, we will add a bias for the year the movie was released.

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

By doing this, our RMSE becomes:

```
## [1] 0.868593
```

While this may look alarming, remember this has not been regularized, so we can expect it to be a bit higher than before.

Regularized Mean + Movie Bias + User Bias + Release Year Bias + Number of Genres Bias Model

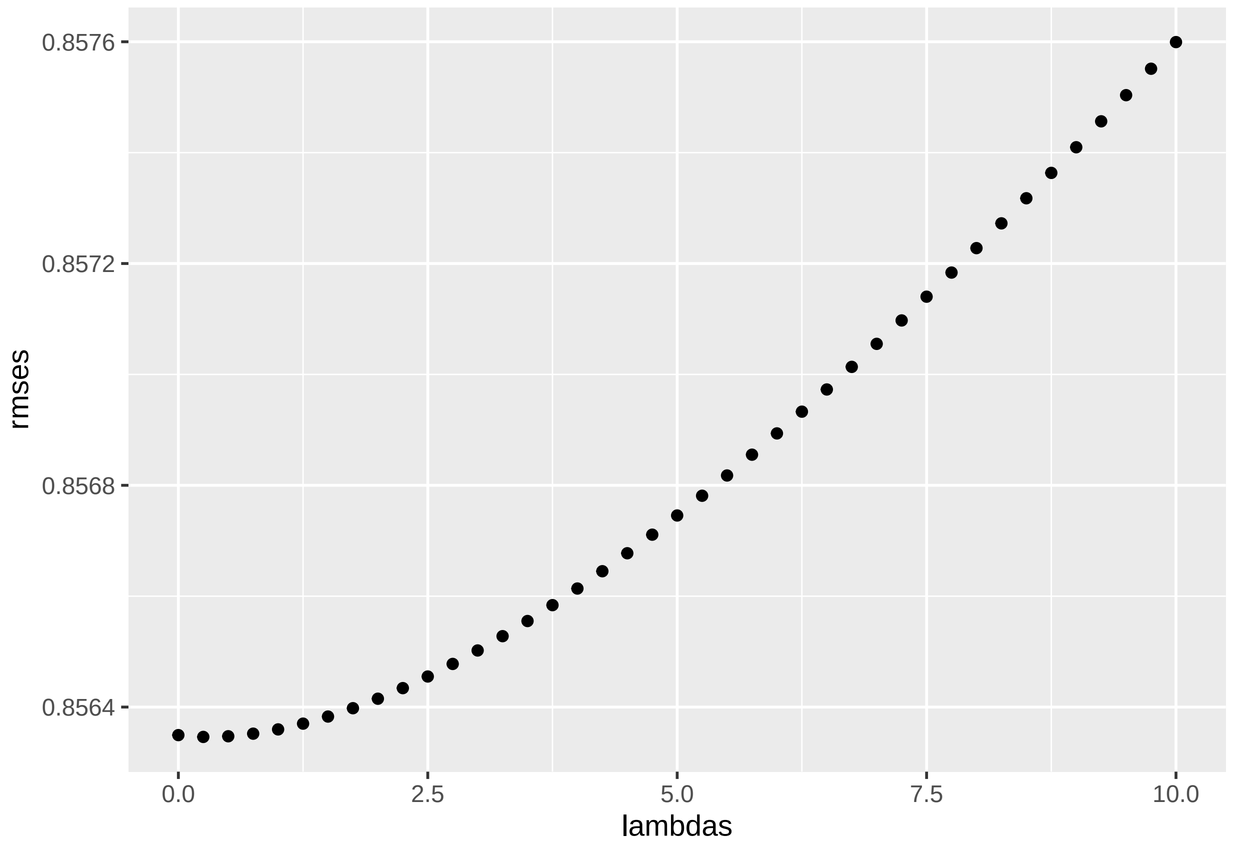
The final term we will add is one for the number of genres a movie includes.

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

## Lambda Optimization

We will also regularize this model, so let's first minimize the RMSE using lambda. We will run through a number of iterations, with the results charted below.

```
lambdas = seq(0, 10, 0.25)
rmses = sapply(lambdas, function(l){
  mu = mean(edx$rating)
  b_i = edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u = edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_y = edx %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(release) %>%
    summarize(b_y = sum(rating - b_i - b_u - mu)/(n()+l))
  b_n = edx %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "release") %>%
    group_by(NUM) %>%
    summarize(b_n = sum(rating - b_i - b_u - mu)/(n() + l))
  predicted_ratings = edx %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "release") %>%
    left_join(b_n, by = "NUM") %>%
    mutate(pred = mu + b_i + b_u + b_y + b_n) %>%
    pull(pred)
return(RMSE(predicted_ratings, edx$rating))
})

qplot(lambdas, rmses)
```



The best lambda was $\lambda = 0.25$.

We will use this in our final model to get the results when used on the test set.

Results

Here is the code for our final model and the RMSE results when used on the test set.

```r
lambda = lambdas[which.min(rmses)]
mu = mean(validation$rating)
b_i = validation %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))
b_u = validation %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
b_y = validation %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by = "userId") %>%
  group_by(release) %>%
  summarize(b_y = sum(rating - b_i - b_u - mu)/(n()+lambda))
b_n = validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_y, by = "release") %>%
  group_by(NUM) %>%
  summarize(b_n = sum(rating - b_i - b_u - mu)/(n() + lambda))
predicted_ratings = validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_y, by = "release") %>%
  left_join(b_n, by = "NUM") %>%
  mutate(pred = mu + b_i + b_u + b_y + b_n) %>%
  pull(pred)
result = RMSE(validation$rating, predicted_ratings)
print(result)
```

```
## [1] 0.8249795
```

Our final result of 0.8249795 is well below our goal of 0.86490.

Through combining regularization, lambda optimization, and adding multiple terms, we were able to bring down the RMSE by over 0.30 points from the Naive Model:

Model	RMSE
Naive Average	1.0603313
Regularized Final	0.8249795

Conclusion

We were able to take a training dataset of 9 million ratings, and constructed our own rating prediction algorithm from scratch that obtained an RMSE score of 0.8249795 on the 1 million observation test set, well below the goal of 0.86490.

One way to improve these scores even more is through matrix factorization. This would be able to include a user's preference for a specific genre. This, however, would necessitate a continuous scoring of how well a movie fit into a given genre.