MovieLens Project

Joseph Wellman

19 May 2020

1. INTRODUCTION

In this project, we will study the MovieLens Dataset - a collection of user ratings of movies covering a period of several years - and will create a movie recommendation system using machine learning techniques.

We will be using the MovieLens 10M Dataset, a subset of the full MovieLens Dataset containing approximately 10 million user ratings, compared to 27 million ratings in the current version of the full set, due to its more manageable size and lower computing time required.

We will first split the MovieLens 10M Dataset into training and validation sets, and will examine the data contained in the training set only. Our goal is to create an algorithm to predict movie ratings in the the validation set by utilizing the data in the training set. Various factors and their relationships to movie ratings in the training dataset will be explored and visualized to determine their relevance for inclusion in our final algorithm.

A machine learning algorithm will then be developed utilizing the training set, which itself will be split into training and test sets. A linear model will be built-up in stages, to measure the effect on performance provided by each incremental factor added. The algorithm's performance will be measured by calculating the Root Mean Square Error (RMSE) when comparing our algorithm's predicted ratings for the test set to the set's actual ratings. We will then look at the effects of regularization on the model's performance and implement cross validation to find the optimal tuning parameter.

Finally, we will test our final optimized algorithm with the validation set and measure the RMSE between our predicted ratings and the actual ratings.

2. DATA ANALYSIS AND ALGORITHM METHODS

2.1 DATA ANALYSIS

We first split the MovieLens 10M Dataset into a training set named `edx` and a test set named `validation`, in approximately a 90:10 ratio. We ensure that movies and users included in the validation set are also included in our training set, so that when running our final algorithm it will not encounter any instances where it has no data available on either a movie or user from which to form a prediction.

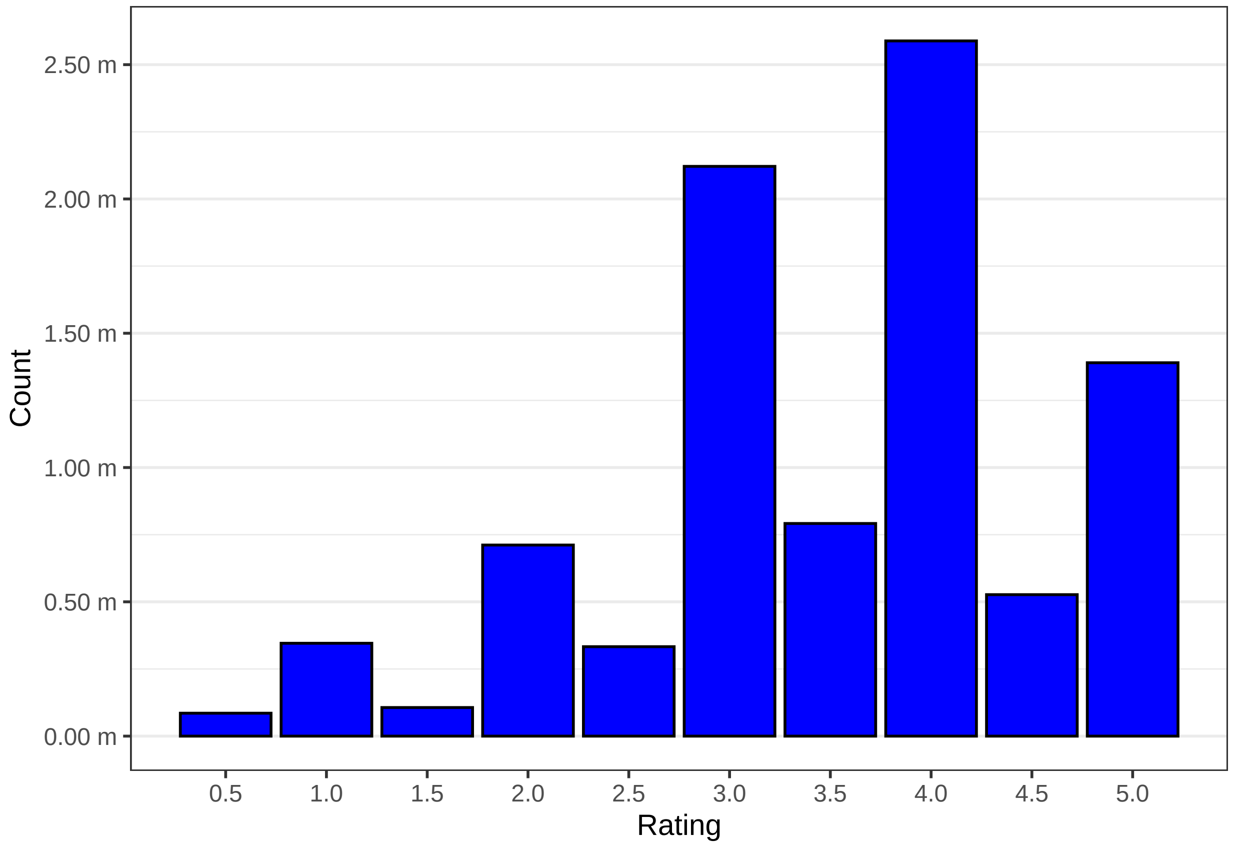
Examining the structure of our training set and the first six rows, we see that it contains just over 9 million entries with six columns:

- `userId` which is unique to each individual user;
- `movieId` which is unique for each movie;
- `rating` giving that user's rating for the movie;
- `timestamp` recording the time and date of the rating;

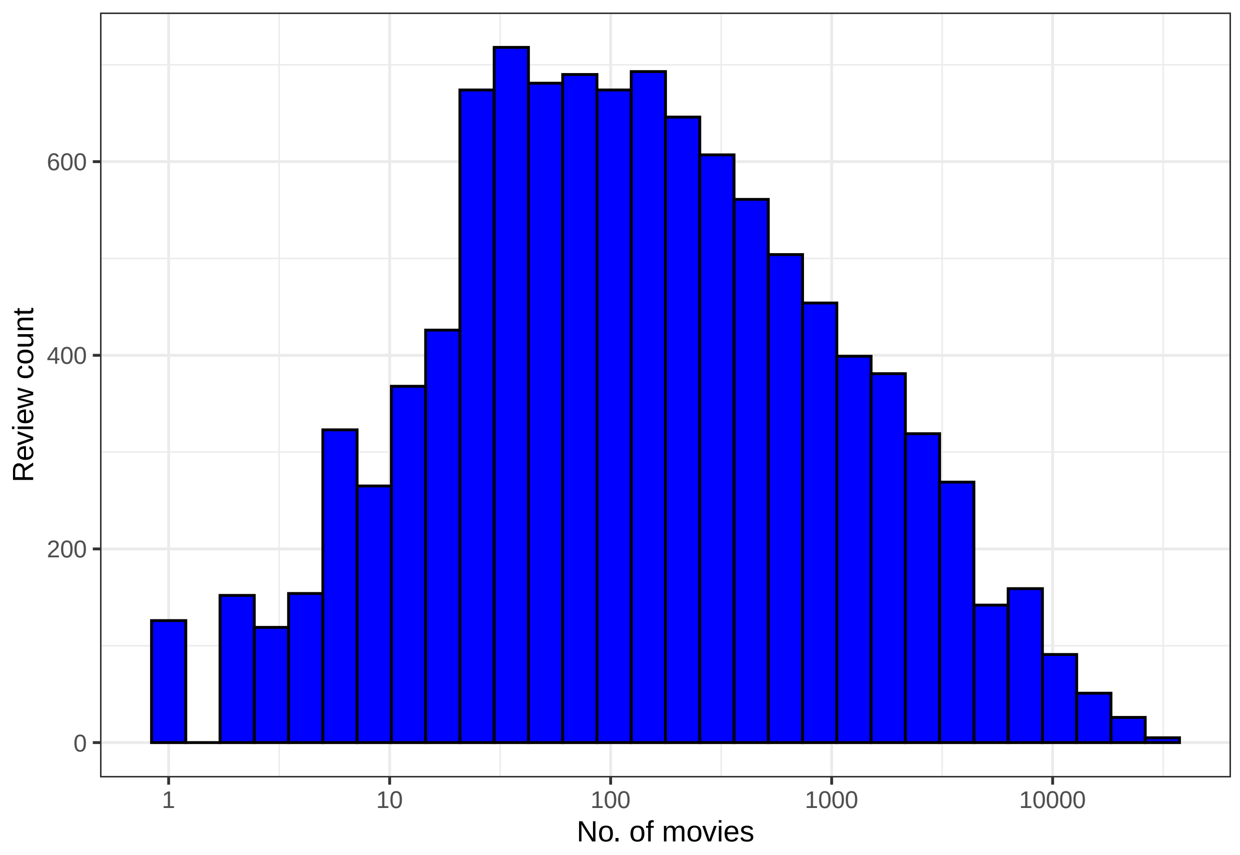
- `title` providing the movie title and the release year in parentheses; and
- `genres` which is a compound description of the genres associated with the movie.

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ac
```

We visualize the distribution of ratings provided by users in a histogram and see that 4.0 and 3.0 are the most common ratings, with the half-step ratings (0.5, 1.5, etc.) being less common than whole number ratings in general.

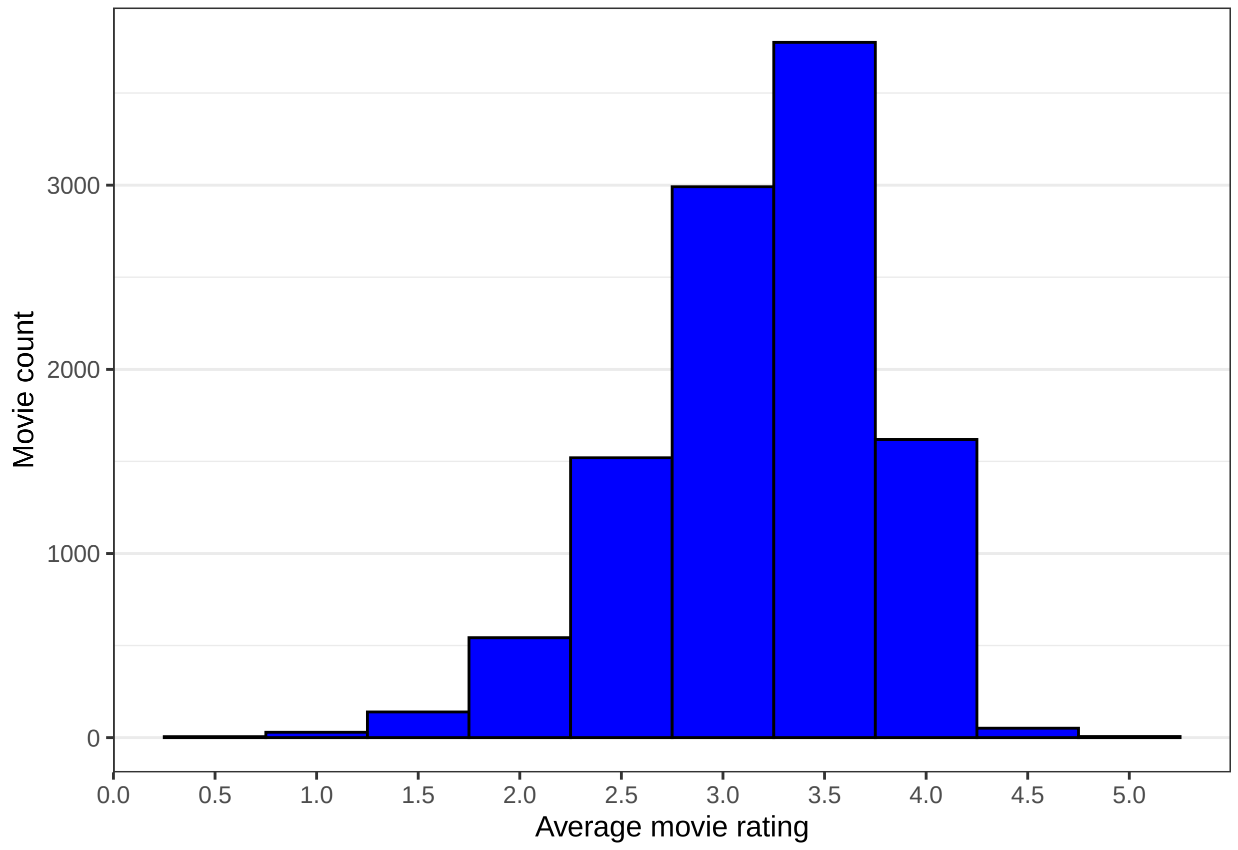


As demonstrated in the data below, some movies have received far more ratings than others:



It can be seen above that the large majority of movies have received less than 100 ratings (categories of 10,000 movies and greater).

Below we plot the distribution of average movie ratings:



```r
mean(edx %>% group_by(title) %>%
  summarize(avg = mean(rating)) %>% .$avg)
```

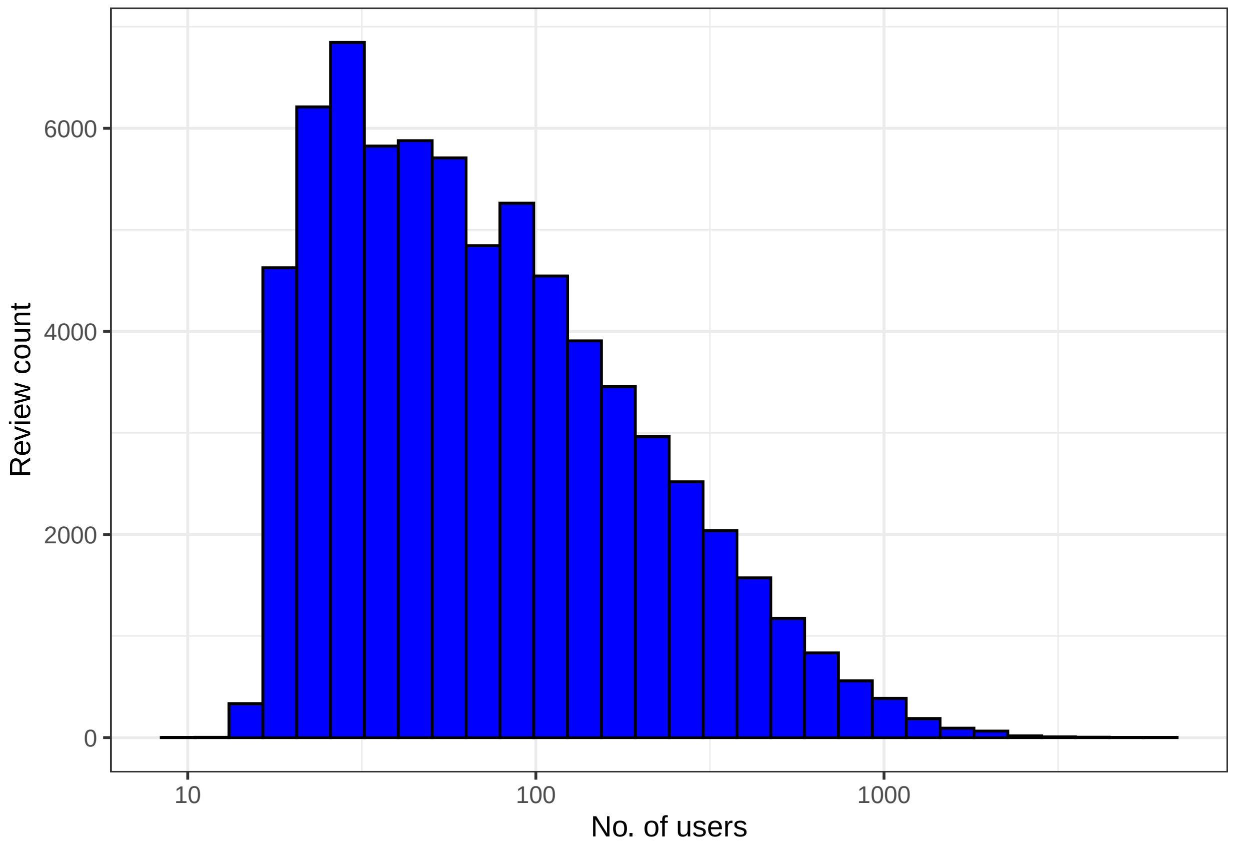
```
## [1] 3.191762
```

```r
sd(edx %>% group_by(title) %>%
  summarize(avg = mean(rating)) %>% .$avg)
```

```
## [1] 0.5713468
```

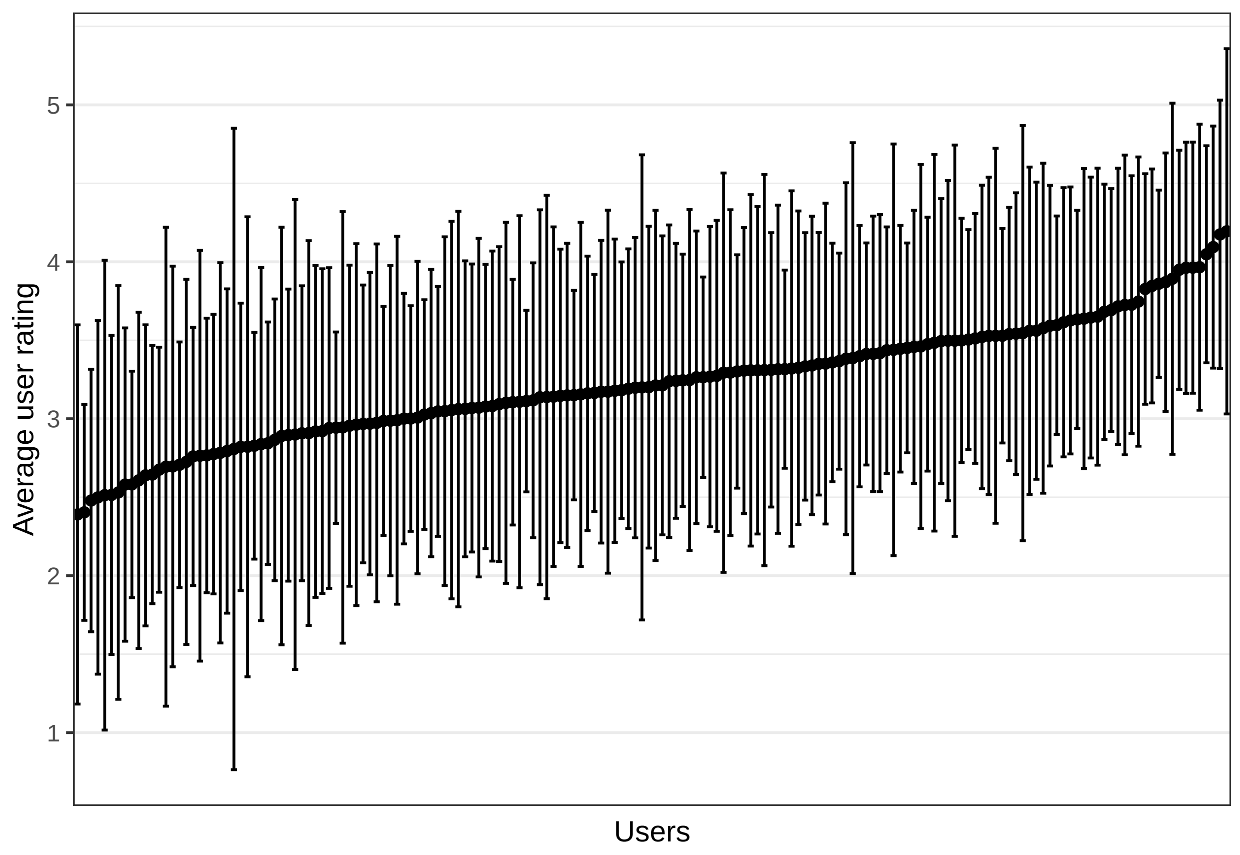
We find that the mean average movie rating is approximately 3.2 with a standard deviation 0.57

Similarly, in the next plot we show the review count by number of users:



We find again that a small number of users have rated far more movies than other users.

We also plot the ordered mean user ratings with error bars at plus and minus one standard deviation for a sample of userId's with over 1,500 reviews:



We can clearly see from the above plot that the variation in the average rating among users is large, as well as the variance for each individual user.

We further examine the number of distinct userId's, movieId's, titles, and genres in our training set:

	Count
No. Reviews	9000055
Distinct Users	69878
Distinct Movies	10677
Distinct Titles	10676
Distinct Genres	797

We see there are 69,878 unique users and 797 distinct compound movie genres, however the number of distinct movieId's and the number of distinct movie titles are off by one: 10,677 versus 10,676. Clearly, each user has only rated a small subset of the movies in our data set since a 'complete' set of ratings would comprise over 746 million entries.

Upon inspection, the movie with the title "War of the Worlds (2005)" is listed under two distinct movieId's:

title	MovieIds
War of the Worlds (2005)	2

One of the movieId's occurs far more frequently than the other, and we also note that some of the genres information is missing for the entries with the second movieId:

movieId	title	count	genres
34048	War of the Worlds (2005)	2460	Action|Adventure|Sci-Fi|Thriller
64997	War of the Worlds (2005)	28	Action

We check that there are no other movie titles associated with this second movieId:

title	movieId	count
War of the Worlds (2005)	64997	28

We thus proceed to amend these 28 entries to have the correct movieId and complete genre information and check our result:

movieId	title	count	genres
34048	War of the Worlds (2005)	2488	Action|Adventure|Sci-Fi|Thriller

We also just check for any other movies with more than one genres description, and see there are no further instances:

```r
edx %>% group_by(title) %>%
  summarize(Genres = n_distinct(genres)) %>%
  filter(Genres > 1) %>%
  knitr::kable()
```

title	Genres

Noticing that the movie release years are contained in the titles, we extract the year information using the `str_extract` function and save this in a title named `years` containing movie titles and their release years.

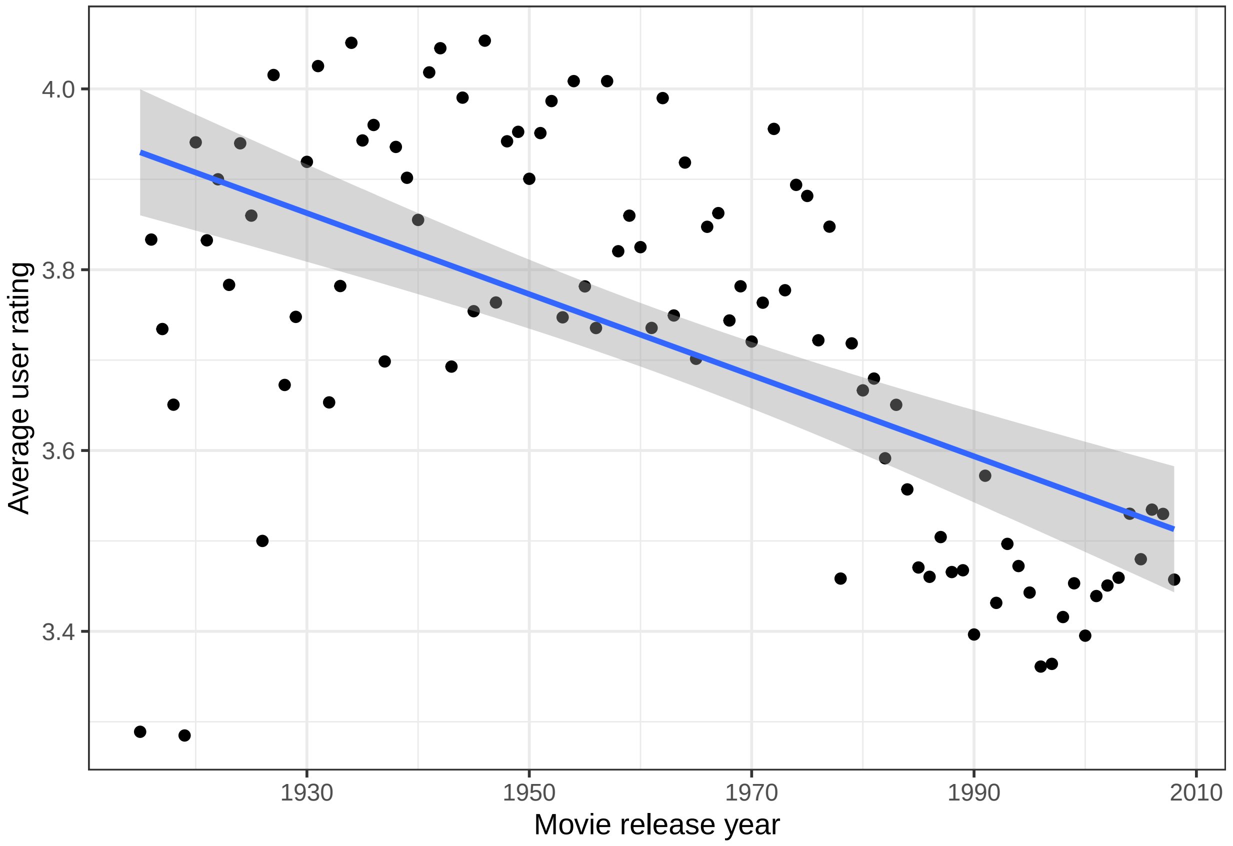
title	year
'burbs, The (1989)	1989
'night Mother (1986)	1986
'Round Midnight (1986)	1986
'Til There Was You (1997)	1997
"Great Performances" Cats (1998)	1998
batteries not included (1987)	1987

We join the `year` column onto our `edx` training set, and will later join it onto the `validation` set during testing:

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

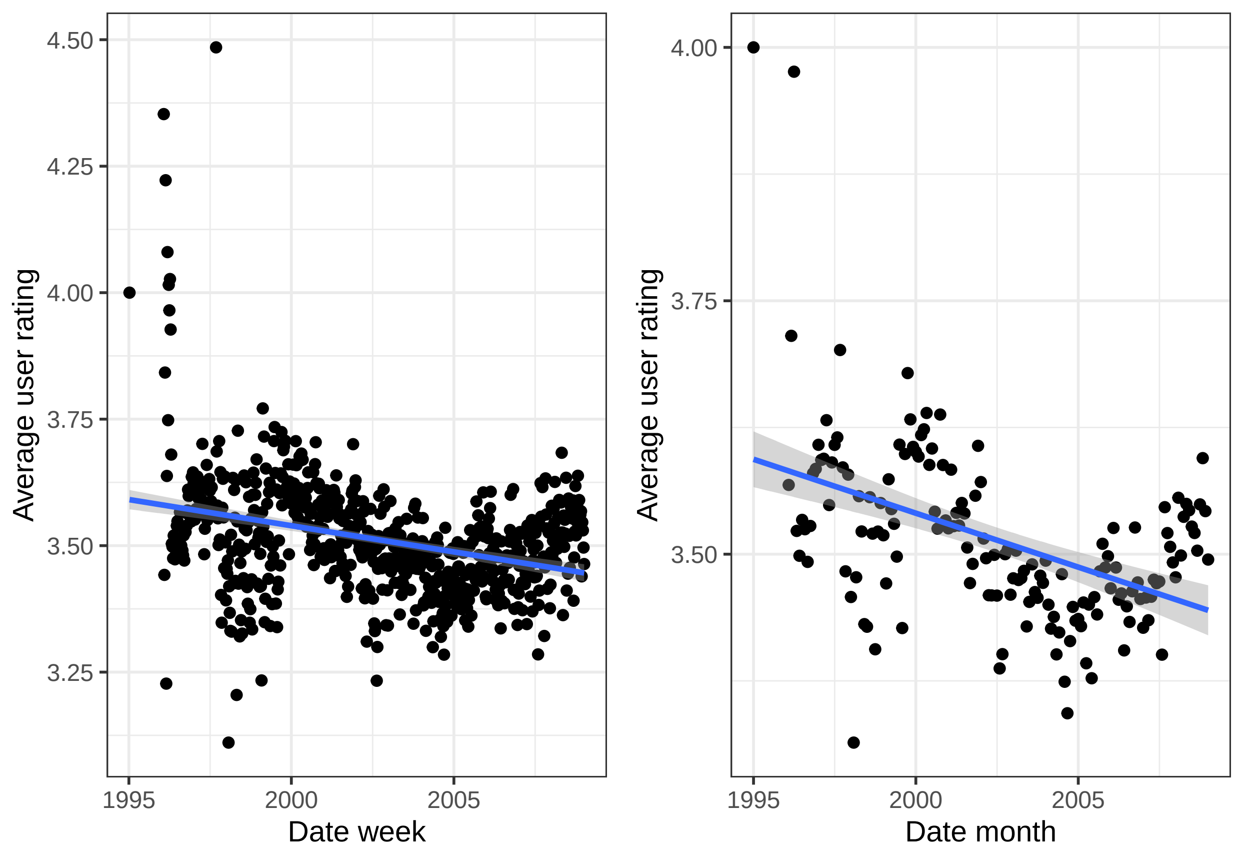
userId	movieId	rating	timestamp	title	genres	year
1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy	1994

We examine the relationship between movie release year and average user rating in the plot below:



There is a noticeable relationship between release year and average rating, with more recent movies receiving lower average ratings.

Now looking at the timestamp data in our training set, we use the `round_date` function in the `lubridate` package to round the timestamp data first to the nearest week, and secondly to the nearest month, and compare the plots of the rounded dates against average ratings during those periods:



From the above we can see a trend whereby average user ratings have declined over time, and that this trend is preserved when utilizing dates rounded to the nearest month rather than by week. We can therefore use the monthly rounded timestamp data in our analysis in order to save on computing time and memory.

We are also interested in the relationship between the popularity of movies among users and their average rating. As a proxy for movie popularity, we use the average number of ratings a movie has received per year since its release year. We note the latest year for reviews in our training set as 2009:

```
max_yr <- year(max(edx$date_mth))
max_yr
```

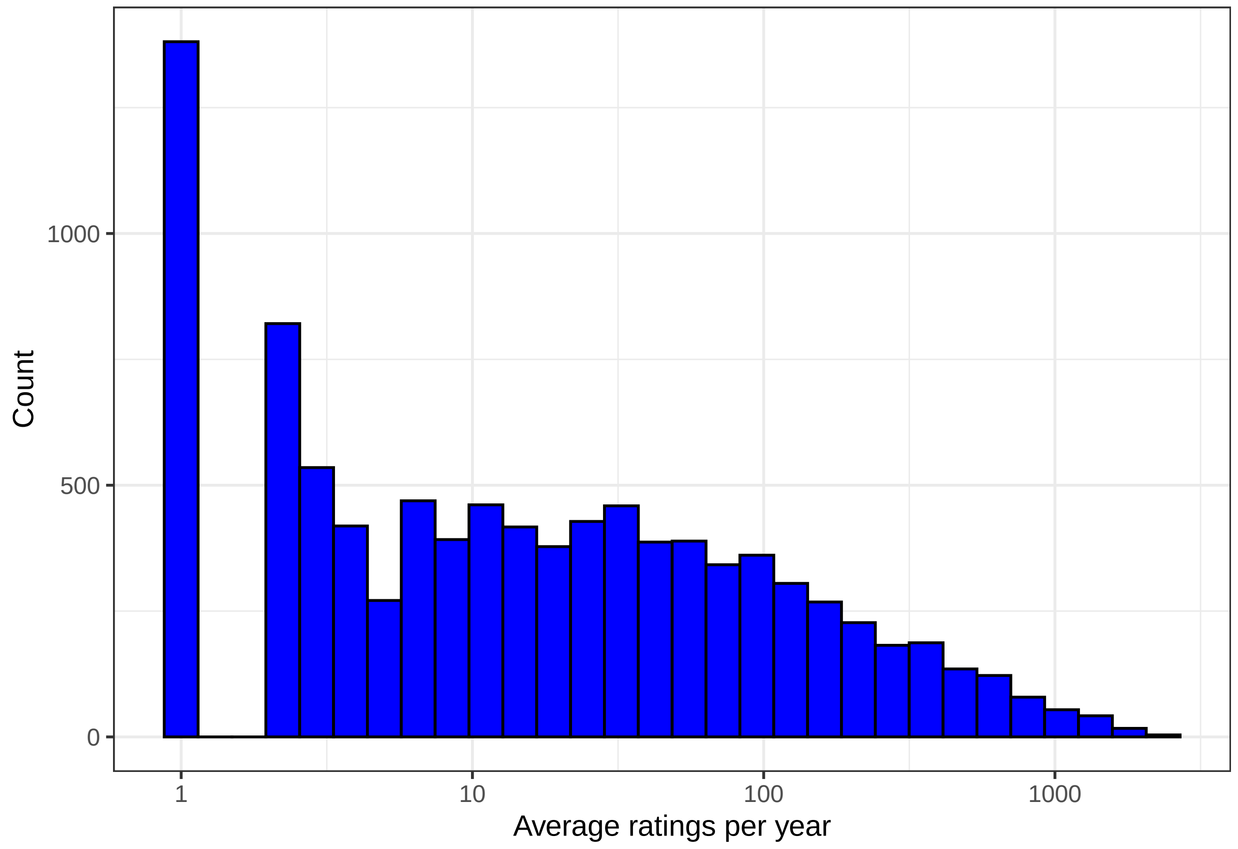
```
## [1] 2009
```

We contruct a data frame named `rtgs_year` which contains movie titles and the rounded average number of ratings received since each movie's release year.

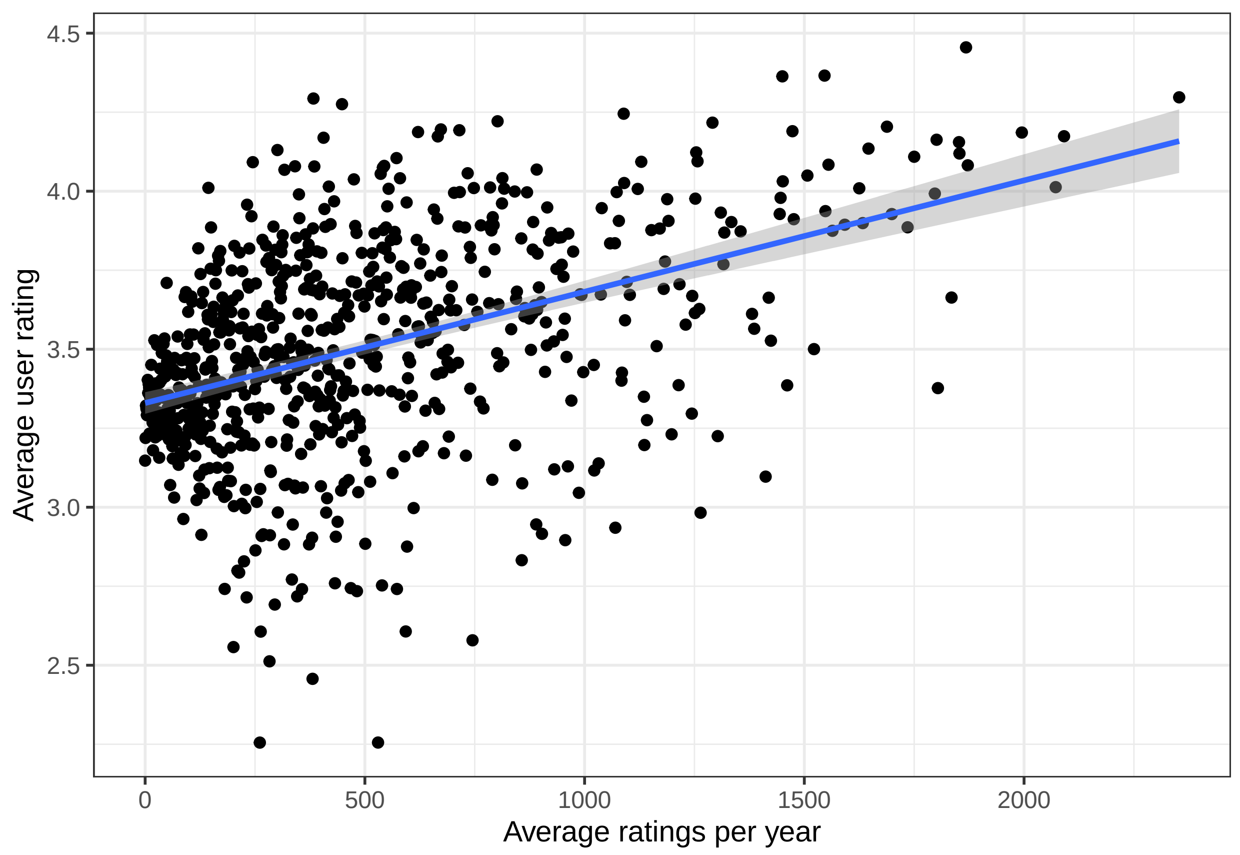
title	rtgs_year
'burbs, The (1989)	67
'night Mother (1986)	9
'Round Midnight (1986)	2
'Til There Was You (1997)	22
"Great Performances" Cats (1998)	0
batteries not included (1987)	20

Looking at the number of distinct entries for ratings per year and the distribution of values, we see that there are 733 different values and they are overwhelmingly skewed toward lower figures:

n_distinct(rtgs_year)
733



Next we plot below the average ratings per year against average user ratings:

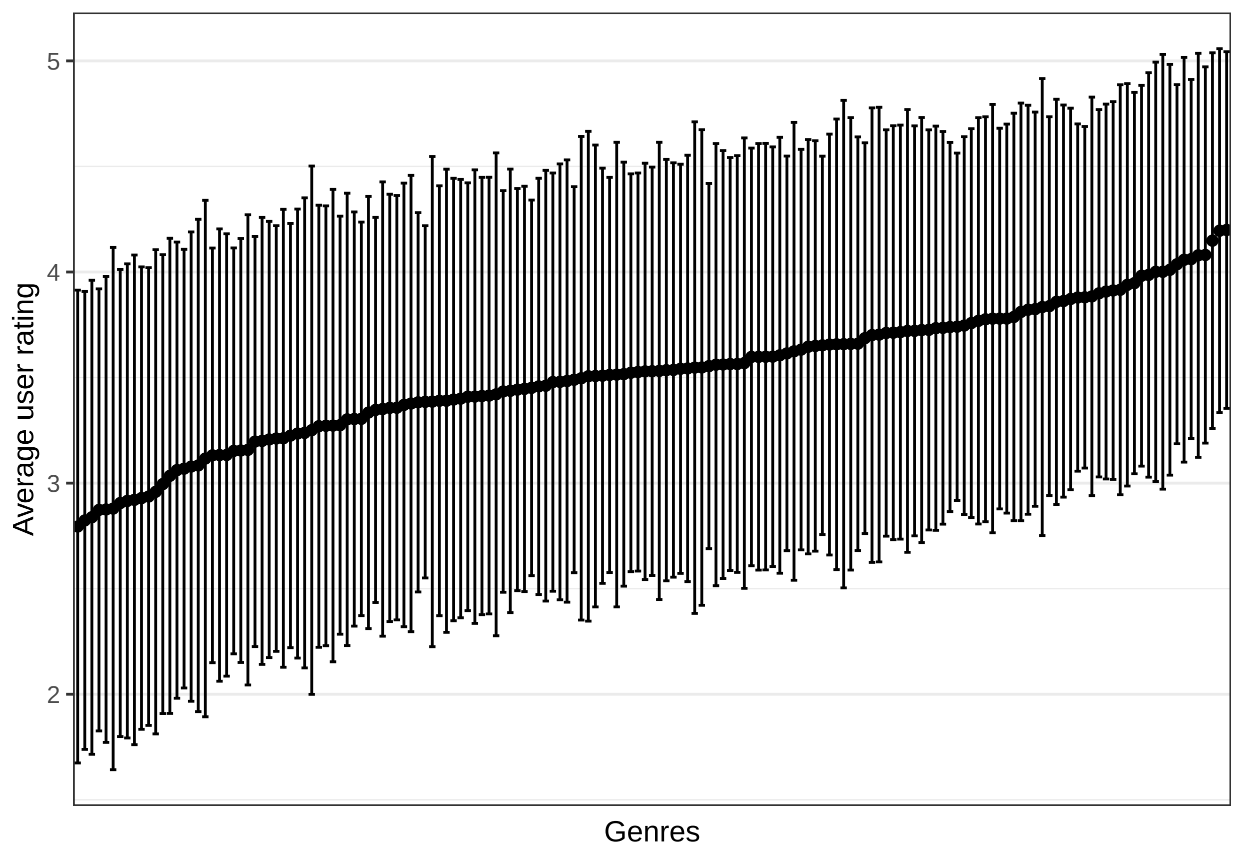


This plot shows that there is a strong positive trend in average user ratings for movies with higher average number of ratings per year, with the vast majority of data points crowded at less than 100.

We join the `rtgs_year` column onto our `edx` training set, and will later join it onto the `validation` set during testing:

userId	movieId	rating	title	genres	year	rtgs_year
1	122	5	Boomerang (1992)	Comedy|Romance	1992	128
1	185	5	Net, The (1995)	Action|Crime|Thriller	1995	962
1	292	5	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	1995	1032
1	316	5	Stargate (1994)	Action|Adventure|Sci-Fi	1994	1135
1	329	5	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	1994	970
1	355	5	Flintstones, The (1994)	Children|Comedy|Fantasy	1994	322

Finally with respect to movie genres information, below we plot the ordered mean ratings with error bars at plus and minus one standard deviation for genres with over 10,000 ratings:



We can see that there is a wide range in the average user ratings across different genres.

We also show the top 10 and bottom 10 genres by rating:

genres	Avg_Rating	Count
Animation|IMAX|Sci-Fi	4.71	7
Action|Crime|Drama|IMAX	4.30	2353
Drama|Film-Noir|Romance	4.30	2989
Animation|Children|Comedy|Crime	4.28	7167
Film-Noir|Mystery	4.24	5988
Crime|Film-Noir|Mystery	4.22	4029
Film-Noir|Romance|Thriller	4.22	2453
Crime|Film-Noir|Thriller	4.21	4844
Action|Adventure|Comedy|Fantasy|Romance	4.20	14809
Crime|Mystery|Thriller	4.20	26892

genres	Avg_Rating	Count
Documentary|Horror	1.45	619
Action|Animation|Comedy|Horror	1.50	2
Action|Horror|Mystery|Thriller	1.61	327
Comedy|Film-Noir|Thriller	1.64	21
Action|Drama|Horror|Sci-Fi	1.75	4
Adventure|Drama|Horror|Sci-Fi|Thriller	1.75	217
Action|Adventure|Drama|Fantasy|Sci-Fi	1.90	57
Action|Children|Comedy	1.91	518

genres	Avg_Rating	Count
Action|Adventure|Children	1.92	824
Adventure|Animation|Children|Fantasy|Sci-Fi	1.92	691

We see in the above that the variance in average ratings for genres is large, ranging from 1.45 up to 4.71, thus providing useful information for our algorithm.

2.2 ALGORITHM METHODS

Before we begin building our algorithm to predict movie ratings, in order to train our algorithm we first split our training set `edx` into a training set containing 80% of entries and a test set containing 20%.

```
## Training set:

## 'data.frame':    7200043 obs. of  9 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  185 316 329 355 364 377 420 539 588 589 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838983525 838983392 838983392 838984474 838983707 838983834 838983834 838984068 83
##  $ title    : chr  "Net, The (1995)" "Stargate (1994)" "Star Trek: Generations (1994)" "Flintstones,
##  $ genres   : chr  "Action|Crime|Thriller" "Action|Adventure|Sci-Fi" "Action|Adventure|Drama|Sci-Fi"
##  $ year     : num  1995 1994 1994 1994 1994 ...
##  $ date_mth : POSIXct, format: "1996-08-01" "1996-08-01" ...
##  $ rtgs_year: num  962 1135 970 322 1261 ...

##
##  Test set:

## 'data.frame':    1799973 obs. of  9 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 2 2 ...
##  $ movieId  : num  122 292 356 362 370 466 520 594 260 376 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 3 ...
##  $ timestamp: int  838985046 838983421 838983653 838984885 838984596 838984679 838984679 838984679 8
##  $ title    : chr  "Boomerang (1992)" "Outbreak (1995)" "Forrest Gump (1994)" "Jungle Book, The (199
##  $ genres   : chr  "Comedy|Romance" "Action|Drama|Sci-Fi|Thriller" "Comedy|Drama|Romance|War" "Adven
##  $ year     : num  1992 1995 1994 1994 1994 ...
##  $ date_mth : POSIXct, format: "1996-08-01" "1996-08-01" ...
##  $ rtgs_year: num  128 1032 2072 241 489 ...
```

We begin with a simple model for our predicted movie ratings Y where we start with the average rating across all movies and users, μ, and add bias terms for movieId, userId, release year and genres. All further differences are explained by an error term:

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

where:

- μ is the average rating across all ratings;
- b_i is the bias term for the average rating of movie i;

- b_u is the bias term for the average rating by user u;
- b_y is the bias term for the average rating for movies released in year y;
- b_g is the bias term for the average rating for movies categorized in genres g; and
- $\epsilon_{i,u,y,g}$ is an error term explaining any further differences.

We could use a linear regression model such as `lm` to fit this model, but again we do not due to the excessive computation time required. We instead compute an approximation by computing $\hat{\mu}$ as the average rating in our training set and then iteratively estimate the bias terms in the following way, where the lower case y terms refer to ratings in our training set:

- $\hat{b}_i$ as the average of $y_i - \hat{\mu}$ for movie i;
- $\hat{b}_u$ as the average of $y_{i,u} - \hat{\mu} - \hat{b}_i$ for movie i, user u;
- $\hat{b}_y$ as the average of $y_{i,u,y} - \hat{\mu} - \hat{b}_i - \hat{b}_u$ for movie i, user u, year y; and
- $\hat{b}_g$ as the average of $y_{i,u,y,g} - \hat{\mu} - \hat{b}_i - \hat{b}_u - \hat{b}_y$ for movie i, user u, year y, genres g.

Model	RMSE
Movie, User, Year + Genres effects model	0.8659566

Our resulting RMSE for the above model is approximately 0.866.

We now look to include an additional bias term, b_d, for the month that the movie rating was made. Our model expands to:

$$Y_{i,u,y,g,d} = \mu + b_i + b_u + b_y + b_g + b_d + \epsilon_{i,u,y,g,d} \ ,$$

where b_d is estimated in the following way:

- $\hat{b}_d$ is the average of $y_{i,u,y,g,d} - \hat{\mu} - \hat{b}_i - \hat{b}_u - \hat{b}_y - \hat{b}_g$ for movie i, user u, year y, genres g, review date d.

Model	RMSE
Movie, User, Year + Genres effects model	0.8659566
Movie, User, Year, Genres + Review Date effects model	0.8658401

We see a slight improvement to our RMSE.

Next, we can also include a bias term relating to movie popularity, or ratings per year, labelled b_p in our model:

$$Y_{i,u,y,g,d,p} = \mu + b_i + b_u + b_y + b_g + b_d + b_p + \epsilon_{i,u,y,g,d,p} \ ,$$

where b_p is estimated as:

- $\hat{b}_p$ is the average of $y_{i,u,y,g,d,p} - \hat{\mu} - \hat{b}_i - \hat{b}_u - \hat{b}_y - \hat{b}_g - \hat{b}_d$ for movie i, user u, year y, genres g, review date d, popularity (ratings per year) p.

Model	RMSE
Movie, User, Year + Genres effects model	0.8659566
Movie, User, Year, Genres + Review Date effects model	0.8658401
Movie, User, Year, Genres, Review Date + Popularity effects model	0.8656037

Again, a marginal improvement is seen in our RMSE.

REGULARIZATION

Finally, we look to include regularization into our model to improve the results. Regularization helps to improve the accuracy of our model by reducing the size of extreme outliers with small sample sizes in our bias estimates.

We seek to minimize the expression:

$$\frac{1}{N}\sum_{i,u,y,g,d,p}(y_{i,u,y,g,d,p}-\mu-b_i-b_u-b_y-b_g-b_d-b_p)^2+\lambda(\sum_i b_i^2+\sum_u b_u^2+\sum_y b_y^2+\sum_g b_g^2+\sum_d b_d^2+\sum_p b_p^2)\ ,$$

where N is the total number of ratings, the first term is a residual sum of squares and the second is a penalty term that is larger when many individual bias terms are large. The term λ is a constant tuning parameter.

The approach for estimating the regularized bias terms is again iterative. We seek our estimate for b_i through minimizing the equation:

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

By using calculus, we find our estimate for b_i for a given movie i:

$$\frac{d}{db_i}\sum_{u=1}^{n_i}(y_{i,u}-\mu-b_i)^2+\lambda b_i^2=0$$

$$\frac{d}{db_i}\sum_{u=1}^{n_i}\left((y_{i,u}-\mu)^2-2(y_{i,u}-\mu)b_i+b_i^2\right)^2+\lambda b_i^2=0$$

$$-2\sum_{u=1}^{n_i}(y_{i,u}-\mu)+2n_ib_i+2\lambda b_i=0$$

$$\hat{b}_i(\lambda)=\frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}(y_{i,u}-\hat{\mu})\ ,$$

where n_i is the number of ratings given for movie i.

Similarly, our estimate of b_u can be calculated by minimizing the equation:

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

Again, through calculus in the same manner it can be proven that the value of b_u which minimizes the above for a user u is the following:

$$\hat{b}_u(\lambda)=\frac{1}{\lambda+n_u}\sum_{i=1}^{n_u}(y_{i,u}-\hat{\mu}-\hat{b_i}),\text{ where } n_u \text{ is the number of ratings given by user } u.$$

It follows that we can estimate the remaining terms in a similar manner through iteration:

b_y for a given release year y may be estimated by:

$$\hat{b}_y(\lambda)=\frac{1}{\lambda+n_y}\sum_{n=1}^{n_y}(y_{i,u,y}-\hat{\mu}-\hat{b_i}-\hat{b_u}),\text{ where } n_y \text{ is the number of ratings for year } y.$$

b_g for given genres g may be estimated by:

$$\hat{b}_g(\lambda) = \frac{1}{\lambda + n_g} \sum_{n=1}^{n_g} (y_{i,u,y,g} - \hat{\mu} - \hat{b_i} - \hat{b_u} - \hat{b_y}), \text{ where } n_g \text{ is the number of ratings for genres } g.$$

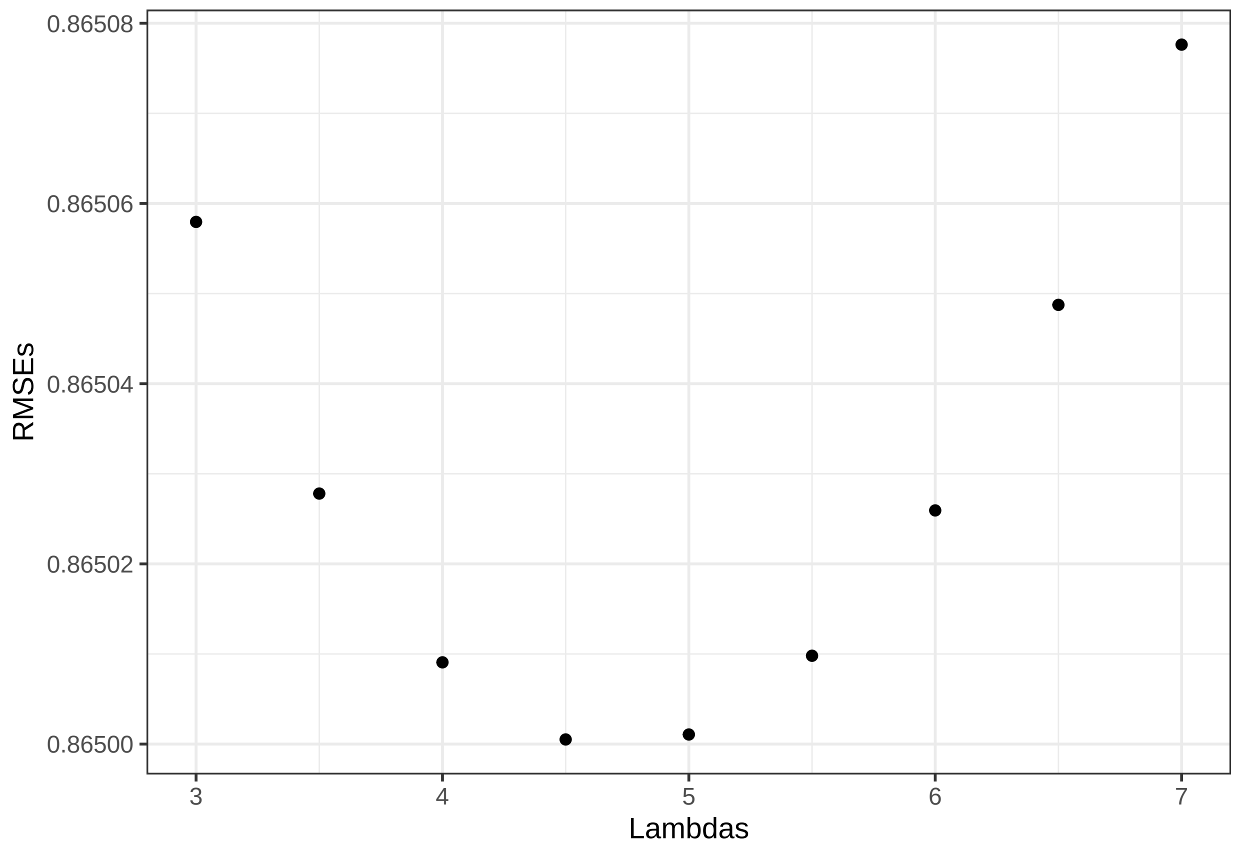
b_d for a given review date d may be estimated by:

$$\hat{b}_d(\lambda) = \frac{1}{\lambda + n_d} \sum_{n=1}^{n_d} (y_{i,u,y,g,d} - \hat{\mu} - \hat{b_i} - \hat{b_u} - \hat{b_y} - \hat{b_g}), \text{ where } n_d \text{ is the number of ratings for review date } d.$$

b_p for a given movie popularity (ratings per year) p may be estimated by:

$$\hat{b}_p(\lambda) = \frac{1}{\lambda + n_p} \sum_{n=1}^{n_p} (y_{i,u,y,g,d,p} - \hat{\mu} - \hat{b_i} - \hat{b_u} - \hat{b_y} - \hat{b_g} - \hat{b_d}), \text{ where } n_p \text{ is the number of ratings for movie popularity } p.$$

We test our algorithm with regularization utilizing estimates for the above bias terms, and use cross-validation to find the optimal value for lambda which minimizes our algorithm's RMSE.



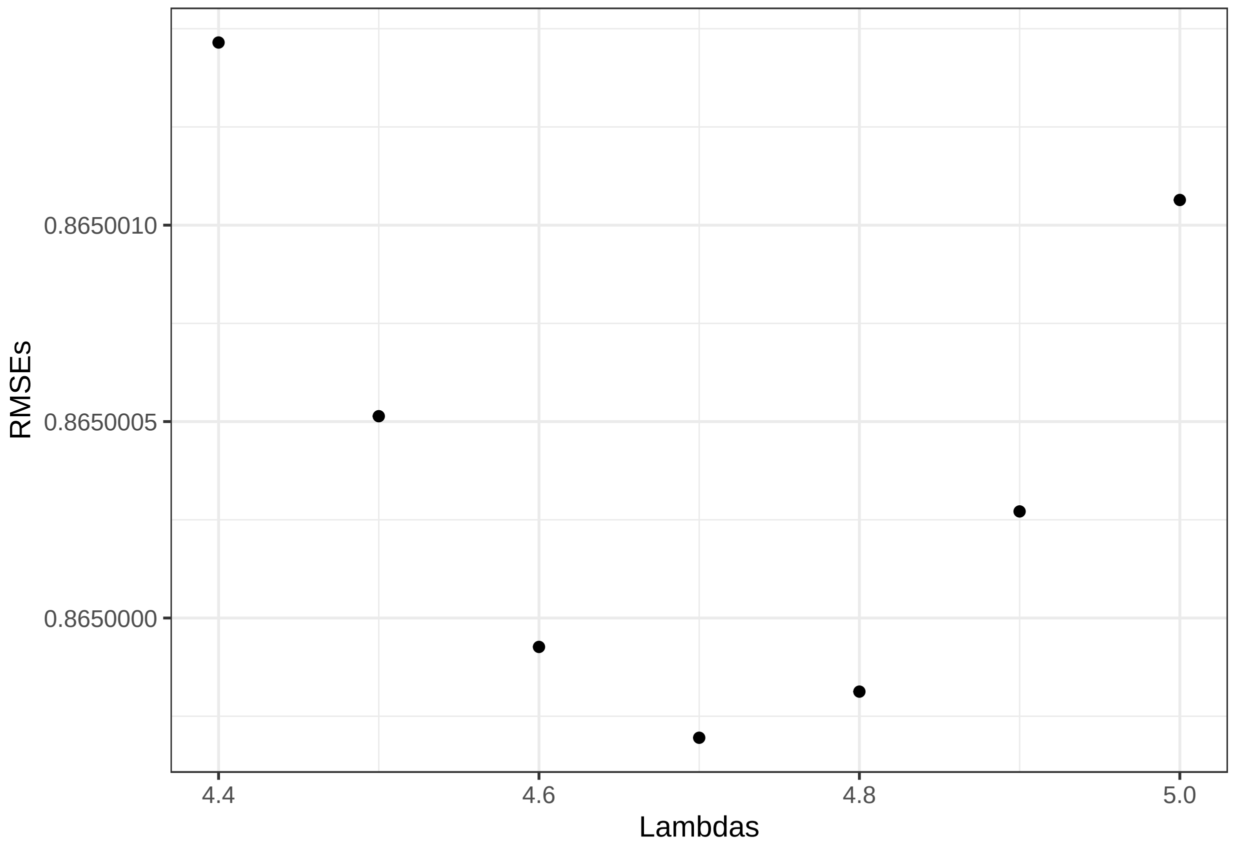
```r
min(rmses)
```

```
## [1] 0.8650005
```

```r
lambdas[which.min(rmses)]
```

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

We see that our best RMSE is approximately 0.865, occurring when lambda equals 4.5. We can try to improve upon this by re-running the cross-validation focusing on a narrower range of values for lambda:



```r
min(rmses)
```

```
## [1] 0.8649997
```

```r
best_lambda <- lambdas[which.min(rmses)]
best_lambda
```

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

Model	RMSE
Movie, User, Year + Genres effects model	0.8659566
Movie, User, Year, Genres + Review Date effects model	0.8658401
Movie, User, Year, Genres, Review Date + Popularity effects model	0.8656037
Regularized model (tuning parameter: 4.7)	0.8649997

We see that our best RMSE result improves to just below 0.865 in the regularized model, which occurs with a lambda value of 4.7. We will proceed to use this model and tuning parameter as our final algorithm to test on the `validation` set.

3. RESULTS

We utilize the regularized model with a tuning parameter of 4.7 to predict the ratings in our `validation` set and then calculate the RMSE compared with the acutal ratings. The mean rating μ and bias terms are calculated using the training `edx` set, the bias terms are then joined onto the validation set together with the movie year and popularity data calculated earlier from the `edx` set, and finally the ratings predictions are calculated as:

$$\hat{Y}_{i,u,y,g,d,p} = \hat{\mu} + \hat{b_i} + \hat{b_u} + \hat{b_y} + \hat{b_g} + \hat{b_d} + \hat{b_p}$$

```
mu <- mean(edx$rating)

b_i <- edx %>%
       group_by(title) %>%
       summarize(b_i = sum(rating - mu)/(n()+best_lambda))

b_u <- edx %>%
       left_join(b_i, by="title") %>%
       group_by(userId) %>%
       summarize(b_u = sum(rating - b_i - mu)/(n()+best_lambda))

b_y <- edx %>%
       left_join(b_i, by="title") %>%
       left_join(b_u, by="userId") %>%
       group_by(year) %>%
       summarize(b_y = sum(rating - b_u - b_i - mu)/(n()+best_lambda))

b_g <- edx %>%
       left_join(b_i, by="title") %>%
       left_join(b_u, by="userId") %>%
       left_join(b_y, by="year") %>%
       group_by(genres) %>%
       summarize(b_g = sum(rating - b_y - b_u - b_i - mu)/(n()+best_lambda))

b_d <- edx %>%
       left_join(b_i, by="title") %>%
       left_join(b_u, by="userId") %>%
       left_join(b_y, by="year") %>%
       left_join(b_g, by="genres") %>%
       group_by(date_mth) %>%
       summarize(b_d = sum(rating - b_g - b_y - b_u - b_i - mu)/(n()+best_lambda))

b_p <- edx %>%
       left_join(b_i, by="title") %>%
       left_join(b_u, by="userId") %>%
       left_join(b_y, by="year") %>%
      left_join(b_g, by="genres") %>%
      left_join(b_d, by="date_mth") %>%
      group_by(rtgs_year) %>%
      summarize(b_p = sum(rating - b_d - b_g - b_y - b_u - b_i - mu)/(n()+best_lambda))


predicted_ratings <- validation %>%
  left_join(years, by = "title") %>%
  left_join(rtgs_year, by = "title") %>%
  mutate(date_mth = round_date(as_datetime(timestamp), unit = "month")) %>%
  left_join(b_i, by = "title") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_y, by = "year") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_d, by = "date_mth") %>%
  left_join(b_p, by = "rtgs_year") %>%
  mutate(pred = mu + b_i + b_u + b_y + b_g + b_d + b_p) %>%
  .$pred

rm(mu, b_i, b_u, b_y, b_g, b_d, b_p)
```

```
RMSE(predicted_ratings, validation$rating)
```

```
## [1] 0.8638947
```

Our final RMSE result for the algorithm is 0.8638947.

4. CONCLUSION

In this project we constructed a machine learning algorithm to predict movie ratings in the MovieLens 10M Dataset. We first split the data set into a training set, `edx`, and a test set, `validation`, in a 90:10 proportion and set about constructing a linear model based on our analysis of the data in the training set.

Upon analyzing the data, we found that movie title, users, release year, review date, movie popularity, and movie genres all looked to have an effect on a given rating. As we then proceeded to construct a linear model utilizing our training set split into its own training and test sets, we confirmed that the incremental addition of all of the above factors had a positive impact to our model since we saw the RMSE fall during testing. We then incorporated regularization into our model and used cross-validation to find the optimal value of the tuning parameter and saw the RMSE fall further.

Finally, when running the final optimized model on our validation set, we find the final RMSE between the predicted and actual ratings to be 0.8638947. This is a strong result, and is a significant improvement on the 0.866 result we initially saw on our training data.

To improve the results further we could look at fitting a multivariate linear regression model using the `lm` function, other models such as a logistic regression, k-Nearest Neighbors, or random forest model, and even multiple methods under an ensemble. Fitting such models may take an excessively long time due to the very large number of data points in the training set. A sample of the training set could be utilized for model fitting purposes, with a trade-off between computing time and model accuracy being made.

In addition, a matrix factorization model using singular value decomposition and principal component analysis could be explored, which could be fitted with the `recommenderlab` package. Again, however, this may take substantial computing time on a data set of this size.