MovieLens Project

Code

Jas Beausejour

March 18, 2019

Note 1: This report was created in the context of the HarvardX Professional Certificate in Data Science program available on edX. It was created solely for the purpose of this course by Jas Beausejour.

Note 2: Running this code can easily take 9-11 minutes. The actual run time is indicated at the end of the report.

Note 3: A barebone R file containing strictly only what is necessary to get to our final RMSE of **0.8649** is also being submitted. However, readers should use this report to see the comments and nuances of the code.

Note 4: This code requires a working internet connection as it downloads data from the web.

Executive Summary

We build a movie recommendation system by using part of the MovieLens database. Our main success metric is the Root Mean Square Error, which we bring down to **0.8649**, within a percentage point of the Netflix Challenge Winners (0.8567).

We do this by creating a model with the following structure:

$$Y_{u,i} = \mu + \hat{b}_i(\lambda_i) + \hat{b}_u(\lambda_u) + \varepsilon_{u,i}$$

$Y_{u,i}$ is the actual rating for a given movie, for a given user. It is the product of μ the average rating in the dataset, $\hat{b}_i(\lambda_u)$ the regularized movie effect, $\hat{b}_u(\lambda_i)$ the regularized user-specific effect and $\varepsilon_{u,i}$ the residual.

The only two tuning parameters of this model are the two λs.

To further improve these results, we suggest using other machine learning algorithms on the remaining $\varepsilon_{u,i}$.

Introduction

Netflix and other streaming and video content providers today are competing on much more than content. Content, to a certain extent, has now become somewhat of a, albeit very expensive, commodity. Indeed, there has never been so much content created every year, yet we do not have more time in our lives to watch said content.

Therefore, the performance of the Movies and TV Shows recommendation algorithms of said content providers is of critical importance.The days of leisurly walking into a BlockBuster store are over, and much of the decision making happens on the spot. We've all been in a situation where we wanted to watch a movie, but couldn't seem to identify one that would fit our tastes on Netflix or Hulu. A personalized and accurate rating prediction algorithm could help solve this issue. If the content provider were able to accurately predict which content we will love, this is a content provider we are likely to hire in the Job to Be Done (see Clayton Christensen's work) of entertaining us That is how video content providers can develop a competitive advantage.

The present project aims to develop such a model from the MovieLens dataset. As mentioned on the course website:

> For this project, [we] will be creating a movie recommendation system using the MovieLens dataset. The version of MovieLens included in the dslabs package (which was used for some of the exercises in PH125.8x: Data Science: Machine Learning) is just a small subset of a much larger dataset with millions of ratings. [We] will be creating [our] own recommendation system using all the tools [we have been shown] throughout the courses in this series. [We] will use the 10M version of the MovieLens dataset to make the computation a little easier.

The main goal of this movie recommendation system will be to minimize the **Root Mean Square Error** of our predictions. The **RMSE** can be computed as follows:

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

We define $y_{u,i}$ as the true rating for movie i by user u and denote our prediction with $\hat{y}_{i,u}$. N is the number of predictions we make in a dataset.

Let us create a formula that automatically calculates the **RMSE**.

Hide

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

Given that all ratings are numbers between 0 and 5, an RMSE of 1 would mean that we are, on average, about 1 star "away from the true rating". We will do our best to outperform this RMSE of 1. To give us a sense of what is excellent, we can turn our attention to the 2009 Netflix Prize contest, which was a similar exercise.

According to Wikipedia:

> On September 18, 2009, Netflix announced team “BellKor’s Pragmatic Chaos” as the prize winner (a Test RMSE of 0.8567), and the prize was awarded to the team in a ceremony on September 21, 2009.

We will be calling this result “Netflix Winner” in our results table. Here we create a results table that we will update throughout this report as our results come in.

Hide

```
best_RMSE <- 0.8567
rmse_results <- data_frame(Method = "Netflix Winner", RMSE = best_RMSE)
kable(rmse_results, align = rep("c",2)) %>%
  kable_styling(full_width = F) %>%
  column_spec(1,bold=T,border_right = T)
```

Method	RMSE
Netflix Winner	0.8567

This report will contain four main sections.

First, we will use the **Creating the Datasets** section as an opportunity to set up the various datasets that we will be using throughout this report. There will be 4 main datasets:

1. **Validation**: This will be our final test set. We will not reference to it until we are ready to test our model and report our final RMSE, on which we will be graded. This will represent 10% of the MovieLens data.
2. **edX**: This is the bulk of the data, on which we will be performing our analysis. It represents about 90% of the MovieLens data. We will act as thought this was this only data available, and therefore split this set further into a train and a test set.
3. **train_set**: This will represent a random set with ~85% of the data contained in the **edX** set, which we will use to train our model.
4. **test_set**: This will represent a random set with ~15% of the data contained in the **edX** set, which we will use to test our model and define which parameters yield the best results. It is based on those results that we will then proceed to test our final model on the **validation** set.

Second, in the **Methods and Analysis** section, we will describe the process we use to create our movie recommendation system, but we will first do a bit of exploratory data analysis to better understand the data set. We will then turn our attention to the modelling of the problem, which we will explain in detail in the hopes that the reader can easily follow our thoughts.

Third, we will use the model created in the previous section to make predictions on the **validation** set. We will then proceed to calculate our final **RMSE**, which will be used for grading purposes. This is the **Results** section.

Fourth, our report will end on a **Conclusion**, where we suggest paths forward to further improve this algorithm.

Creating the Datasets

First, we run the code supplied by the edX staff. This will create for us the **edX** and **validation** set. To shorten this report, we do not display the code here.

Next, we create our **train_set** and **test_set** data sets.

Hide

```
# Test set will be 15% of edX data
set.seed(1)
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.15, list = FALSE)
train_set <- edx[-test_index,]
temp_set <- edx[test_index,]
# Make sure userId and movieId in test set are also in train set
test_set <- temp_set %>%
     semi_join(train_set, by = "movieId") %>%
     semi_join(train_set, by = "userId")
# Add rows removed from test set set back into train set
removed_rows <- anti_join(temp_set, test_set)
train_set <- rbind(train_set, removed_rows)
rm(removed_rows, temp_set, test_index)
```

We are now ready to get started.

Methods and Analysis

In this section, we will start by exploring our data sets, do some visualization, and then start modelling our movie prediction system until we are satisfied with our RMSE.

Visualization and Data Exploration

First, let us take a look at how many rows and variables each data set has:

- **validation**: 999999 rows by 6 variables
- **edX**: 9000055 rows by 6 variables
- **train_set**: 7650081 rows by 6 variables
- **test_set**: 1349974 rows by 6 variables

Unsurprisingly, the sum of rows in **train_set** and **test_set** equal the number of rows in the **edX** dataset. Our code worked well.

Let's quickly look at what variables are available.

Hide

```
names(validation)
```

```
[1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

- **userId**: This is a unique identifier for each individual user.
- **movieId**: This is a unique identifier for each individual movie. We will use this rather than the title because character variables are much more prone to typos.
- **rating**: This is the rating given by a particular user for a particular movie. It is also what we will be trying to predict.
- **Timestamp**: This is a record of *when* that movie was rated by the user.
- **Title**: This is the title of the movie which was rated.
- **Genres**: This is a complex charater variable that lists the various genres pertaining to a particular movie.

For instance, for record 234-238 in the validation set we have.

Hide

```
kable(validation[234:238,], align = rep("l",6)) %>%
  kable_styling(full_width = F) %>%
  column_spec(1,bold=T,border_right = T)
```

	userId	movieId	rating	timestamp	title	genres
234	19	594	5	877081092	Snow White and the Seven Dwarfs (1937)	Animation|Children|Drama|Fantasy|Musical
235	19	596	4	877093423	Pinocchio (1940)	Animation|Children|Fantasy|Musical
236	19	709	3	877087451	Oliver & Company (1988)	Adventure|Animation|Children|Comedy|Musical
237	19	802	4	877080821	Phenomenon (1996)	Drama|Romance
238	19	899	4	877081935	Singin' in the Rain (1952)	Comedy|Musical|Romance

Let's look at the distribution of ratings given in the **edx** set.

Hide

```
ratings_freqency <- edx %>%
  group_by(rating) %>%
  count() %>%
  rename("Rating"=rating, "Frequency"=n) %>%
  mutate("Of total"=paste(round(100*Frequency/nrow(edx),2),"%",sep=""))
kable(ratings_freqency, align = rep("c",3)) %>%
  kable_styling(full_width = F) %>%
  column_spec(1,bold=T,border_right = T)
```

Rating	Frequency	Of total
0.5	85374	0.95%
1.0	345679	3.84%
1.5	106426	1.18%

Rating	Frequency	Of total
2.0	711422	7.9%
2.5	333010	3.7%
3.0	2121240	23.57%
3.5	791624	8.8%
4.0	2588430	28.76%
4.5	526736	5.85%
5.0	1390114	15.45%

We can see that the two most frequent ratings are 4 and 3. It would also seem like half-notes are quite rare. Let's verify this hypothesis.

Hide

```
half_vs_full_ratings <- edx %>%
  mutate("Type"=ifelse(rating %in% c(1,2,3,4,5),"Full","Half")) %>%
  group_by(Type) %>%
  select(Type) %>%
  count() %>%
  rename("Ratings"=n) %>%
  mutate("Of total"=paste(round(100*Ratings/nrow(edx),2),"%",sep=""))
kable(half_vs_full_ratings, align = rep("c",2)) %>%
  kable_styling(full_width = F) %>%
  column_spec(1,bold=T,border_right = T)
```

Type	Ratings	Of total
Full	7156885	79.52%
Half	1843170	20.48%

Indeed, full ratings are ~80% of all ratings given.

The **edx set** is unique on every row, but the number of movies represented is not equal to the number of rows. To count the number of unique movies represented in the set, we use this quick piece of code. Note that we are using the **movieId** variable in case there might be some typos in the **title** variable.

Hide

```
length(unique(edx$movieId))
```

```
[1] 10677
```

We can make the same analysis for the number of unique users.

Hide

```
length(unique(edx$userId))
```

```
[1] 69878
```

Now, let's see which movies have the greatest number of ratings. We expect those to be well-known titles.

Hide

```
movies_by_number_of_rankings <- edx %>%
  group_by(title) %>%
  count() %>%
  ungroup() %>%
  arrange(-n) %>%
  rename("Movie"=title, "Ratings"=n)
kable(head(movies_by_number_of_rankings), align = rep("c",2)) %>%
  kable_styling(full_width = F) %>%
  column_spec(1,bold=T,border_right = T)
```

Movie	Ratings
Pulp Fiction (1994)	31362
Forrest Gump (1994)	31079
Silence of the Lambs, The (1991)	30382
Jurassic Park (1993)	29360
Shawshank Redemption, The (1994)	28015
Braveheart (1995)	26212

Of course, we are in a dataset where not every user has rated every movie. As a matter of fact, those "blanks" are in some ways what we are striving to predict! Let's see an example where we show the ratings given by UserIds 13 through 25 for the 5 movies with the most ratings.

Hide

```
keep <- edx %>%
  count(movieId) %>%
  top_n(5, n) %>%
  .$movieId
tab <- edx %>%
  filter(movieId%in%keep) %>%
  filter(userId %in% c(13:25)) %>%
  select(userId, title, rating) %>%
  mutate(title = str_remove(title, ", The"),
         title = str_remove(title, ":.*")) %>%
  spread(title, rating)
kable(tab) %>%
  kable_styling(full_width = F) %>%
  column_spec(1,bold=T,border_right = T)
```

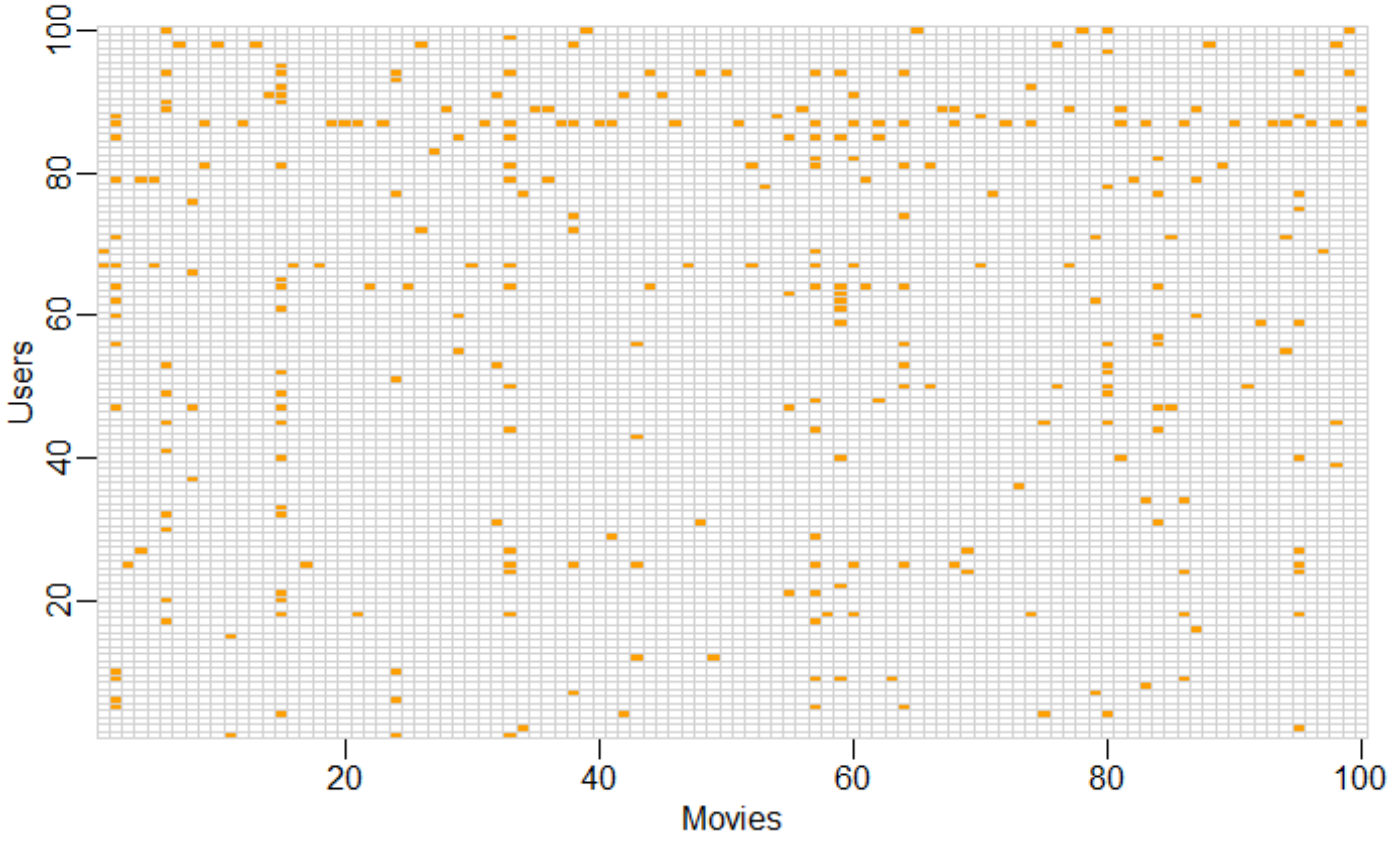
userId	Forrest Gump (1994)	Jurassic Park (1993)	Pulp Fiction (1994)	Shawshank Redemption (1994)	Silence of the Lambs (1991)
13	NA	NA	4	NA	NA
16	NA	3	NA	NA	NA
17	NA	NA	NA	NA	5
18	NA	3	5	4.5	5
19	4	1	NA	4.0	NA
22	NA	4	5	NA	5
23	NA	NA	4	NA	5

We see, for instance, that userID 13 has only rated Pulp Fiction. userID 18 has been much more active, rating every movie except Forrest Gump.

We can get a sense of the sparcity of the data by looking at a matrix for a random sample of 100 movies and 100 users with yellow indicating a user/movie combination for which we have a rating. As we can see here, the data is very sparse.

Hide

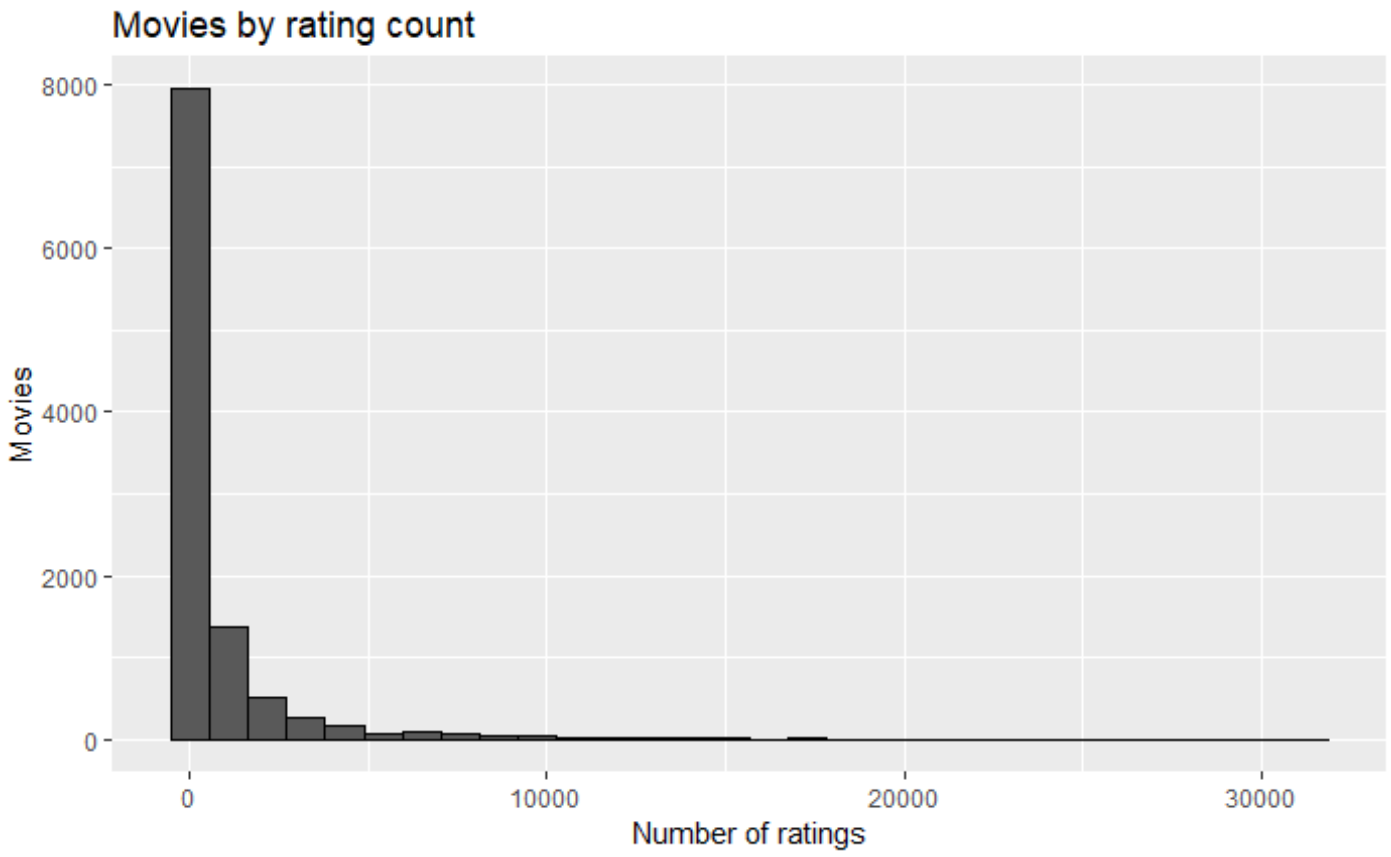
```
if(!require(rafalib)) install.packages("rafalib", repos = "http://cran.us.r-project.org")
library(rafalib)
users <- sample(unique(edx$userId), 100)
rafalib::mypar()
edx %>% filter(userId %in% users) %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>% select(sample(ncol(.), 100)) %>%
  as.matrix() %>% t(.) %>%
  image(1:100, 1:100,. , xlab="Movies", ylab="Users") %>%
  abline(h=0:100+0.5, v=0:100+0.5, col = "lightgrey")
```



Of course, some movies receive a lot more ratings than others. This is not surprising as blockbusters are expected to be rated more frequently than niche movies. We can get a sense of the distribution.

Hide

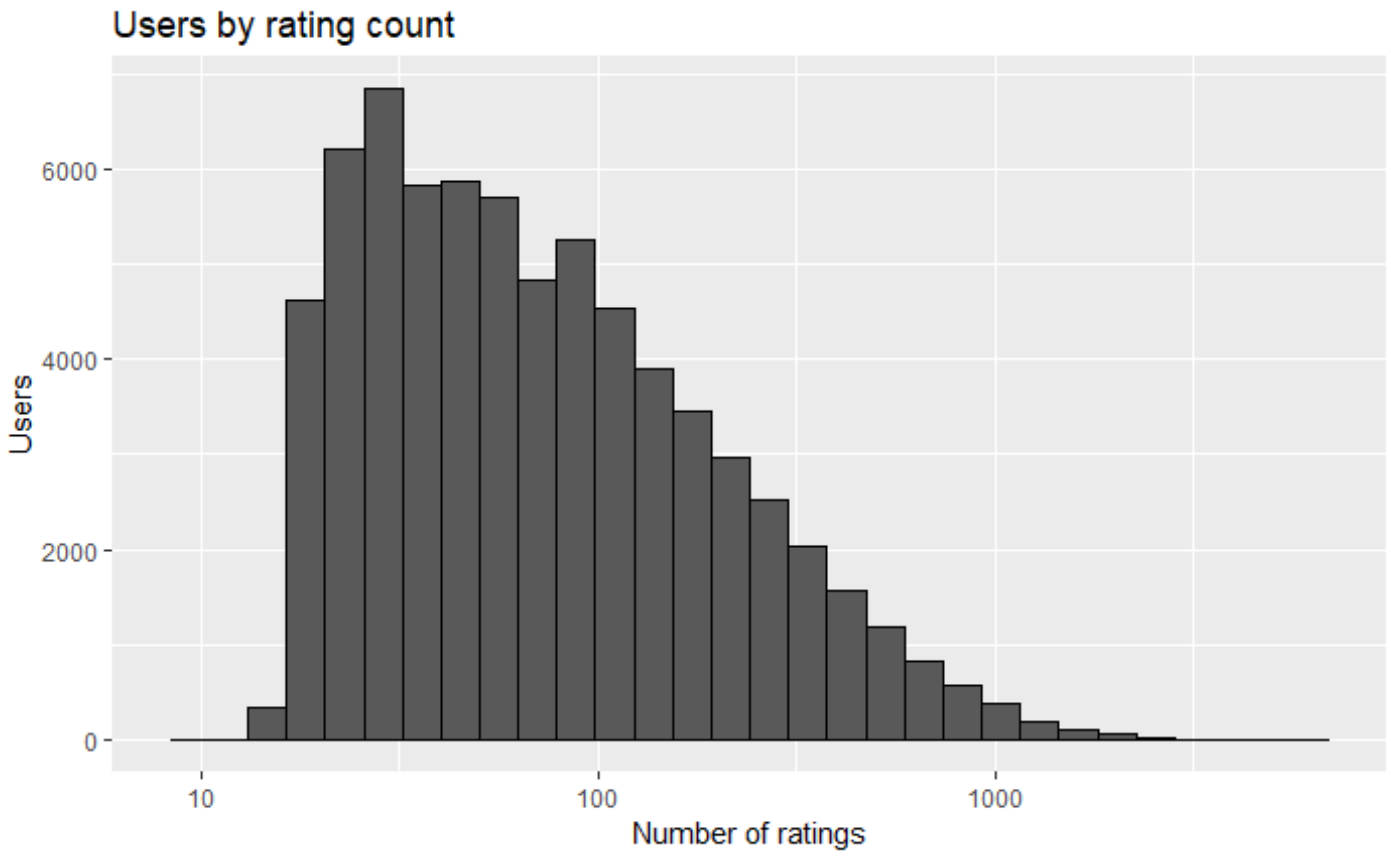
```
edx %>%
  count(movieId) %>%
  arrange(-n) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  labs(title="Movies by rating count", y="Movies", x="Number of ratings")
```



Of course, the same can be said about users: some are much more active than others.

Hide

```
edx %>%
  count(userId) %>%
  arrange(-n) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10()+
  labs(title="Users by rating count", y="Users", x="Number of ratings")
```



Let’s now identify whether different users tend to rate movies more harshly than others

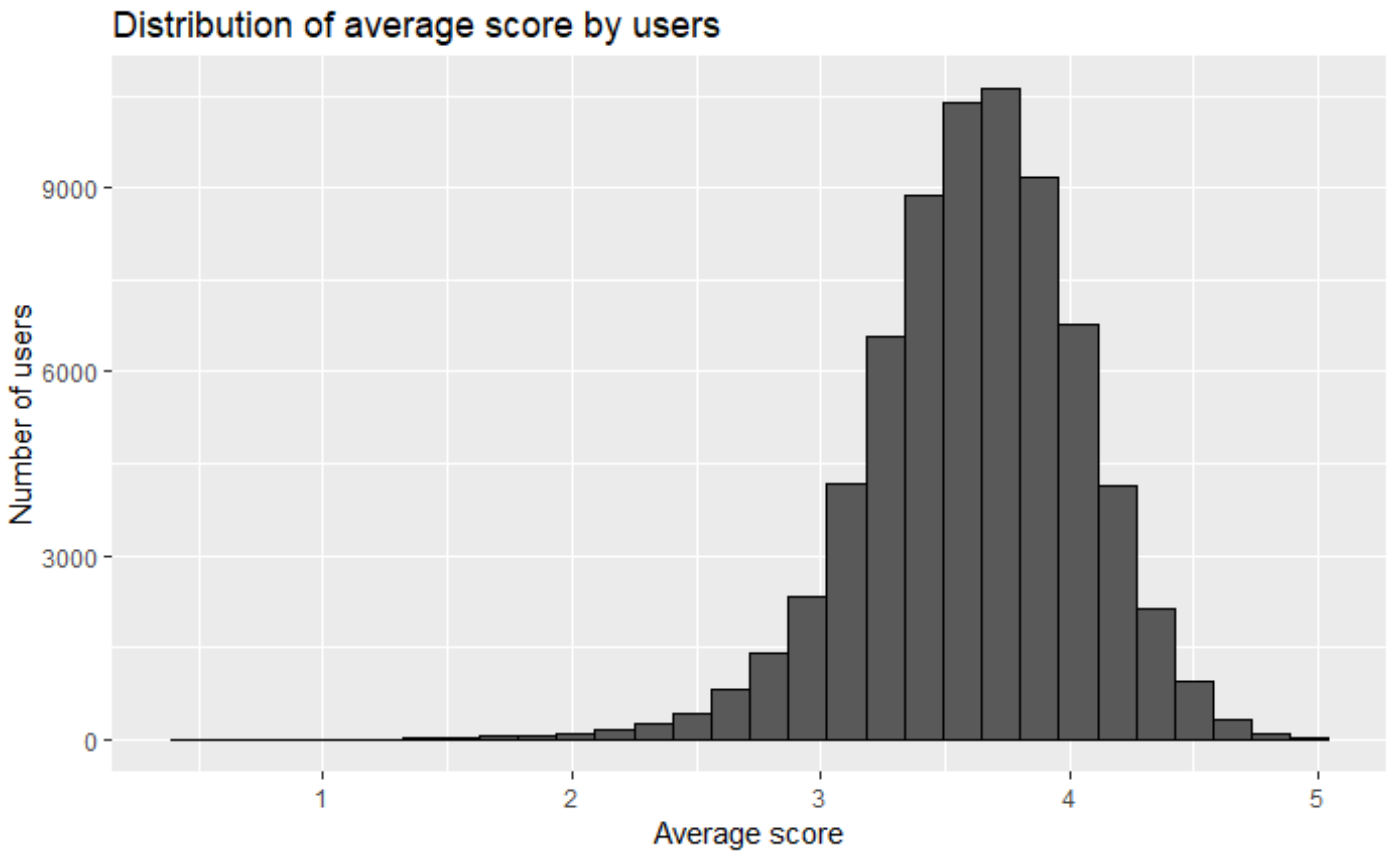
Hide

```
Avg_by_user <- edx %>%
  group_by(userId) %>%
  summarize(avg=mean(rating))
```

On average, users give an average score of 3.6136016. That being said, there is indeed a distribution around this mean.

Hide

```
Avg_by_user %>% ggplot(aes(avg)) +
  geom_histogram(bins = 30, color = "black")+
  labs(title="Distribution of average score by users", x="Average score", y="Number of users")
```



We now feel like we have a pretty good understanding of the data. We are facing a very sparse matrix in which some movies received a lot more ratings than others. On the flip side, some users have given a lot more ratings than others. We will need to take that into consideration in our model.

In addition, we now know that the average rating in the **edX** dataset is 3.5124652. We know that 4 and 3 are the most common ratings and that half ratings are relatively rare.

Modelling

Our methodology will be very similar to that taught in the course. Slowly but surely, we will build to a model with the following equation.

$$Y_{u,i} = \mu + \hat{b}_i(\lambda_i) + \hat{b}_u(\lambda_u) + \varepsilon_{u,i}$$

Here, $Y_{u,i}$ is the actual rating for a given movie, for a given user. It is the product of μ the average rating in the dataset, $\hat{b}_i(\lambda_i)$ the regularized movie effect, $\hat{b}_u(\lambda_u)$ the regularized user-specific effect and $\varepsilon_{u,i}$ the residual.

We could also re-write this equation.

$$Y_{u,i} = \hat{Y_{u,i}} + \varepsilon_{u,i}$$

This shows that our prediction $\hat{Y_{u,i}}$ plus the residual equals the actual rating. Since the residuals represent the errors of our predictions, we will be striving to minimize $\varepsilon_{u,i}$.

The above is consistent with our RMSE formula:

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

Average Rating

Let's start by making predictions where all predictions are the average rating μ. The model would then be:

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

Hide

```
average_rating <- mean(train_set$rating)
naiveRMSE <- RMSE(test_set$rating, average_rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method = "Just the average", RMSE = naiveRMSE))
kable(rmse_results,align=rep("c",3)) %>%
  kable_styling(full_width = F) %>%
  column_spec(1,bold=T,border_right = T)
```

Method	RMSE
Netflix Winner	0.856700
Just the average	1.060077

Of course, here we can see that our RMSE is quite bad.

Movie Effect

It is logical to think that, one average, some movies will be rated better, on average, by all users. We would call this the **movie effect**. For each movie, the movie effect is calculated as the average of $Y_{u,i} - \hat{\mu}$ for each movie i.

We calculate it using this piece of code:

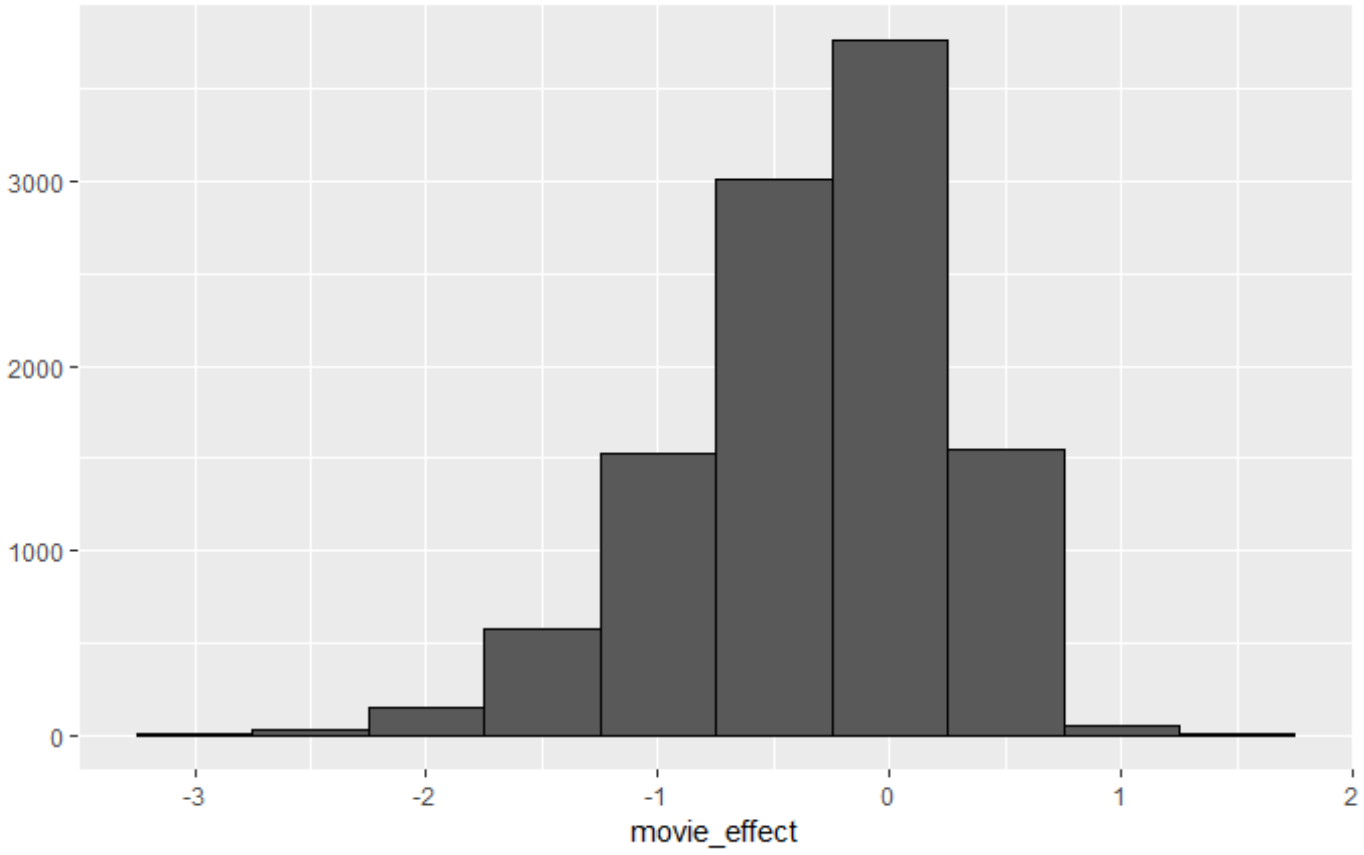
Hide

```
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(movie_effect = mean(rating - average_rating))
```

We can see that these estimates vary substantially:

Hide

```
movie_avgs %>% qplot(movie_effect, geom ="histogram", bins = 10, data = ., color = I("black"))
```



Since our average rating μ is about 3.5, a 1.5 movie_effect b_i means a perfect score of 5.

We can now try to estimate each observation of the test set using this movie_effect. To be clear, this is the model we are trying, where μ is the average rating, b_i is the movie_effect and $\varepsilon_{u,i}$ is the error term:

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

Hide

```
predicted_ratings <- average_rating + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$movie_effect
movie_effect_RMSE <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method= "With Movie Effect",
                                     RMSE=movie_effect_RMSE))
kable(rmse_results,align=rep("c",3), caption = "Metrics calculated on test set only") %>%
  kable_styling(full_width = F) %>%
  column_spec(1,bold=T,border_right = T)
```

Metrics calculated on test set only

Method	RMSE
Netflix Winner	0.856700
Just the average	1.060077

Method	RMSE
With Movie Effect	0.944017

This already has provided a nice improvement from 1.0600772 to 0.944017. We are still quite far away from our "Netflix Winner" however.

Regularizing the Movie Effect

At this point, we make the hypothesis that some of the large movie effects in our data set are due to a very low number of ratings being available for certain movies. For instance, one movie might have been rated only once and given 5 stars. It would therefore have a very high b_i of ~1.5. There is a case to be made that this shouldn't be the case as we have limited information regarding that movie.

First, let's create a database that connects `movieId` to movie title:

Hide

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

Here are the 10 best movies according to our estimate and how often they were rated:

Hide

```
train_set %>% count(movieId) %>%
  left_join(movie_avgs, by ="movieId") %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(movie_effect)) %>%
  select(title, movie_effect,n) %>%
  rename("Title"=title,"Movie Effect"=movie_effect,"Times Rated"=n) %>%
  slice(1:10) %>%
  kable() %>%
  kable_styling(full_width = F) %>%
  column_spec(1,bold=T,border_right = T)
```

Title	Movie Effect	Times Rated
Hellhounds on My Trail (1999)	1.487526	1
Satan's Tango (SÃ¡tÃ¡ntangÃ³) (1994)	1.487526	2
Shadows of Forgotten Ancestors (1964)	1.487526	1
Fighting Elegy (Kenka erejii) (1966)	1.487526	1
Sun Alley (Sonnenallee) (1999)	1.487526	1
Blue Light, The (Das Blaue Licht) (1932)	1.487526	1
More (1998)	1.404192	6

Title	Movie Effect	Times Rated
Human Condition II, The (Ningen no joken II) (1959)	1.237526	4
Constantine's Sword (2007)	1.237526	2
I'm Starting From Three (Ricomincio da Tre) (1981)	1.154192	3

And here are the 10 worst:

Hide

```
train_set %>% count(movieId) %>%
  left_join(movie_avgs, by ="movieId") %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(movie_effect) %>%
  select(title, movie_effect,n) %>%
  rename("Title"=title,"Movie Effect"=movie_effect,"Times Rated"=n) %>%
  slice(1:10) %>%
  kable() %>%
  kable_styling(full_width = F) %>%
  column_spec(1,bold=T,border_right = T)
```

Title	Movie Effect	Times Rated
Besotted (2001)	-3.012474	1
Hi-Line, The (1999)	-3.012474	1
Accused (Anklaget) (2005)	-3.012474	1
Confessions of a Superhero (2007)	-3.012474	1
War of the Worlds 2: The Next Wave (2008)	-3.012474	2
SuperBabies: Baby Geniuses 2 (2004)	-2.756660	43
Disaster Movie (2008)	-2.650405	29
From Justin to Kelly (2003)	-2.648838	176
Hip Hop Witch, Da (2000)	-2.603384	11
Criminals (1996)	-2.512474	1

Our hypothesis was right: we hare heavily benefiting or penalizing obscure movies when we actually have very little data points for them. These are noisy estimates that we should not trust, especially when it comes to predictions. Large errors can increase our RMSE, so we would rather be conservative when unsure.

We will now estimate the **regularized_movie_effect** such that:

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

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

The intuition here is that as n_i gets bigger, the impact of adding λ diminishes. However, for small values of n_i, the presence of λ reduces the estimate of the movie effect $\hat{b}_i(\lambda)$.

Let's compute these regularized estimates of **regularized_movie_effect** using $\lambda = 3$. Later, we will obtimize this term.

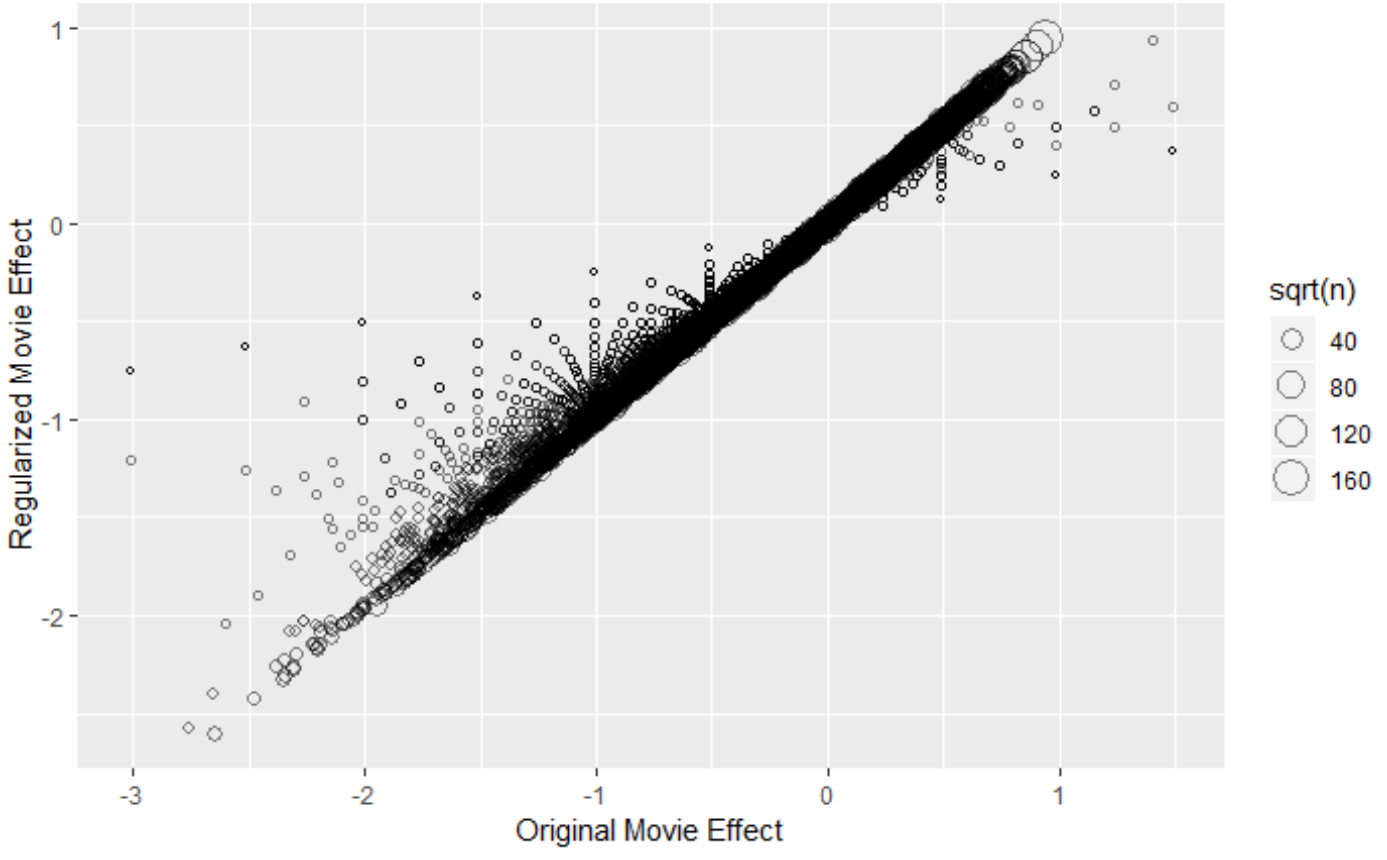
Hide

```
lambda <- 3
movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(reg_movie_effects = sum(rating - average_rating)/(n()+lambda), n_i = n())
```

To see how the estimates shrink, let's make a plot of the regularized estimates versus the least squares estimates.

Hide

```
data_frame("Original Movie Effect" = movie_avgs$movie_effect,
           "Regularized Movie Effect" = movie_reg_avgs$reg_movie_effects,
           n = movie_reg_avgs$n_i) %>%
    ggplot(aes(`Original Movie Effect`, `Regularized Movie Effect`, size=sqrt(n))) +
        geom_point(shape=1, alpha=0.5)
```



Let's look at the top 10 best movies based on $\hat{b}_i(\lambda)$:

Hide

```
train_set %>% count(movieId) %>%
  left_join(movie_reg_avgs, by ="movieId") %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(reg_movie_effects)) %>%
  select(title, reg_movie_effects,n) %>%
  rename("Title"=title,"Reg. Movie Effect"=reg_movie_effects,"Times Rated"=n) %>%
  slice(1:10) %>%
  kable() %>%
  kable_styling(full_width = F) %>%
  column_spec(1,bold=T,border_right = T)
```

Title	Reg. Movie Effect	Times Rated
Shawshank Redemption, The (1994)	0.9439265	23781
More (1998)	0.9361282	6
Godfather, The (1972)	0.9036475	15072
Usual Suspects, The (1995)	0.8537817	18334
Schindler's List (1993)	0.8523260	19749
Rear Window (1954)	0.8112919	6735
Casablanca (1942)	0.8078388	9569
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	0.8043544	2489
Double Indemnity (1944)	0.7931014	1851
Seven Samurai (Shichinin no samurai) (1954)	0.7924817	4383

This makes a lot more sense. These movies are widely known as some of the best ever made.

Let's see if this improves our results further:

Hide

```
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  mutate(pred = average_rating + reg_movie_effects) %>%
  .$pred
reg_movie_effect_RMSE <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method= "With Regularized Movie Effect",
                                     RMSE=reg_movie_effect_RMSE))
kable(rmse_results,align=rep("c",3), caption = "Metrics calculated on test set only") %>%
  kable_styling(full_width = F) %>%
  column_spec(1,bold=T,border_right = T)
```

Metrics calculated on test set only

Method	RMSE
Netflix Winner	0.8567000
Just the average	1.0600772
With Movie Effect	0.9440170
With Regularized Movie Effect	0.9439867

We can see that the improvement from regularization, although quite sensical, is not very large.

This might be due to the fact that we arbitrarily selected $\lambda = 3$.

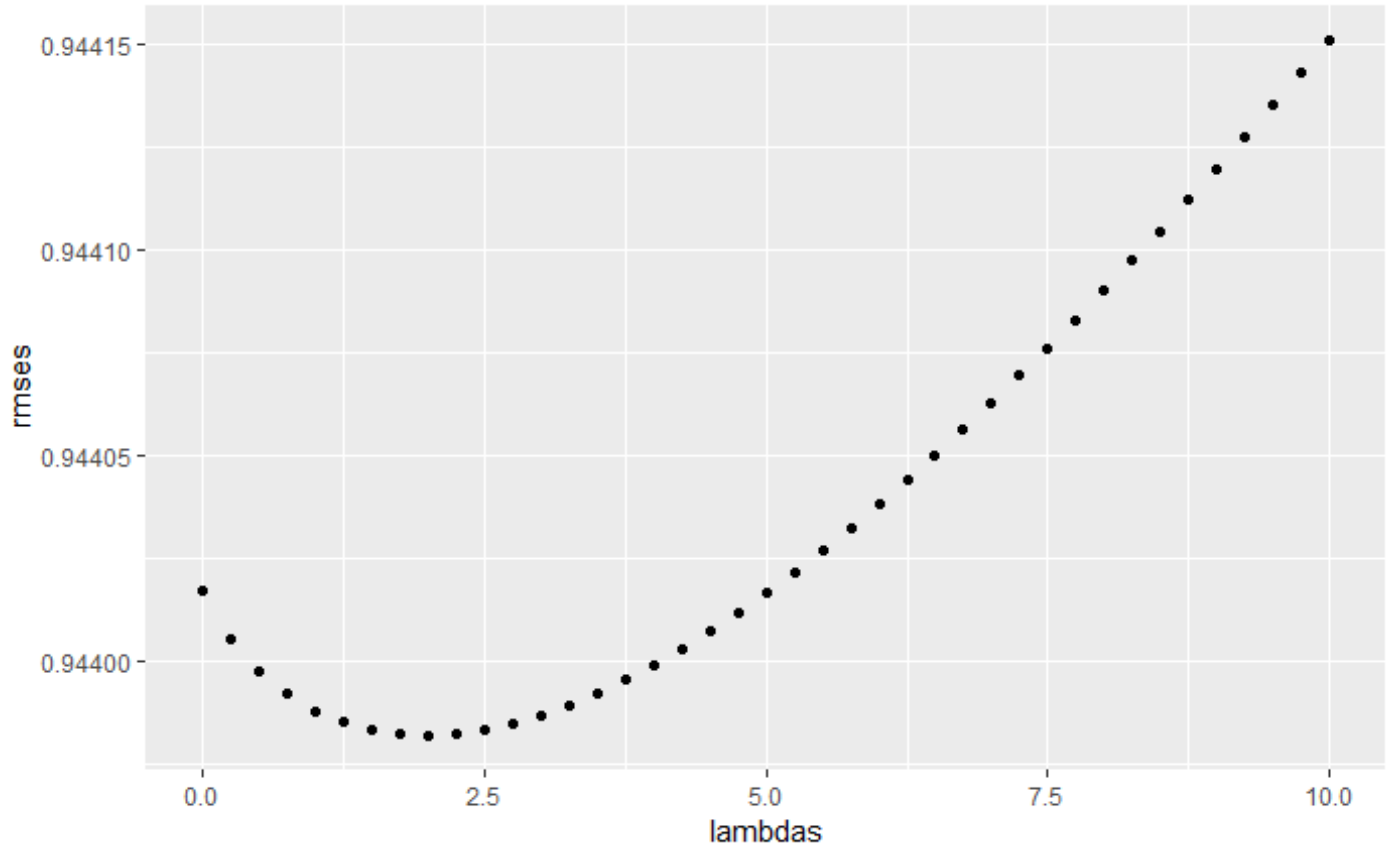
Optimizing Lambda for the Movie Effect

Because λ is a tuning parameter, we can use cross-validation to choose it.

Note that here we are using our test set to perform tuning. This is not best practice, but since the final evaluation will be on the validation set, which we do not use here, we are comfortable doing so.

Hide

```
lambdas <- seq(0, 10, 0.25)
mu <- mean(train_set$rating)
just_the_sum <- train_set %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test_set %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})
qplot(lambdas, rmses)
```



Therefore, for the best results, we should use this lambda.

Hide

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

```
[1] 2
```

Let's now use this optimized lambda to see how it improves our results.

Hide

```
lambda <- lambda_movie
movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(reg_movie_effects = sum(rating - average_rating)/(n()+lambda), n_i = n())
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  mutate(pred = average_rating + reg_movie_effects) %>%
  .$pred
reg_movie_effect_RMSE <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method= "With Optimized Regularized Movie Effect",
                                     RMSE=reg_movie_effect_RMSE))
kable(rmse_results,align=rep("c",3), caption = "Metrics calculated on test set only") %>%
  kable_styling(full_width = F) %>%
  column_spec(1,bold=T,border_right = T)
```

Metrics calculated on test set only

Method	RMSE
Netflix Winner	0.8567000
Just the average	1.0600772
With Movie Effect	0.9440170
With Regularized Movie Effect	0.9439867
With Optimized Regularized Movie Effect	0.9439818

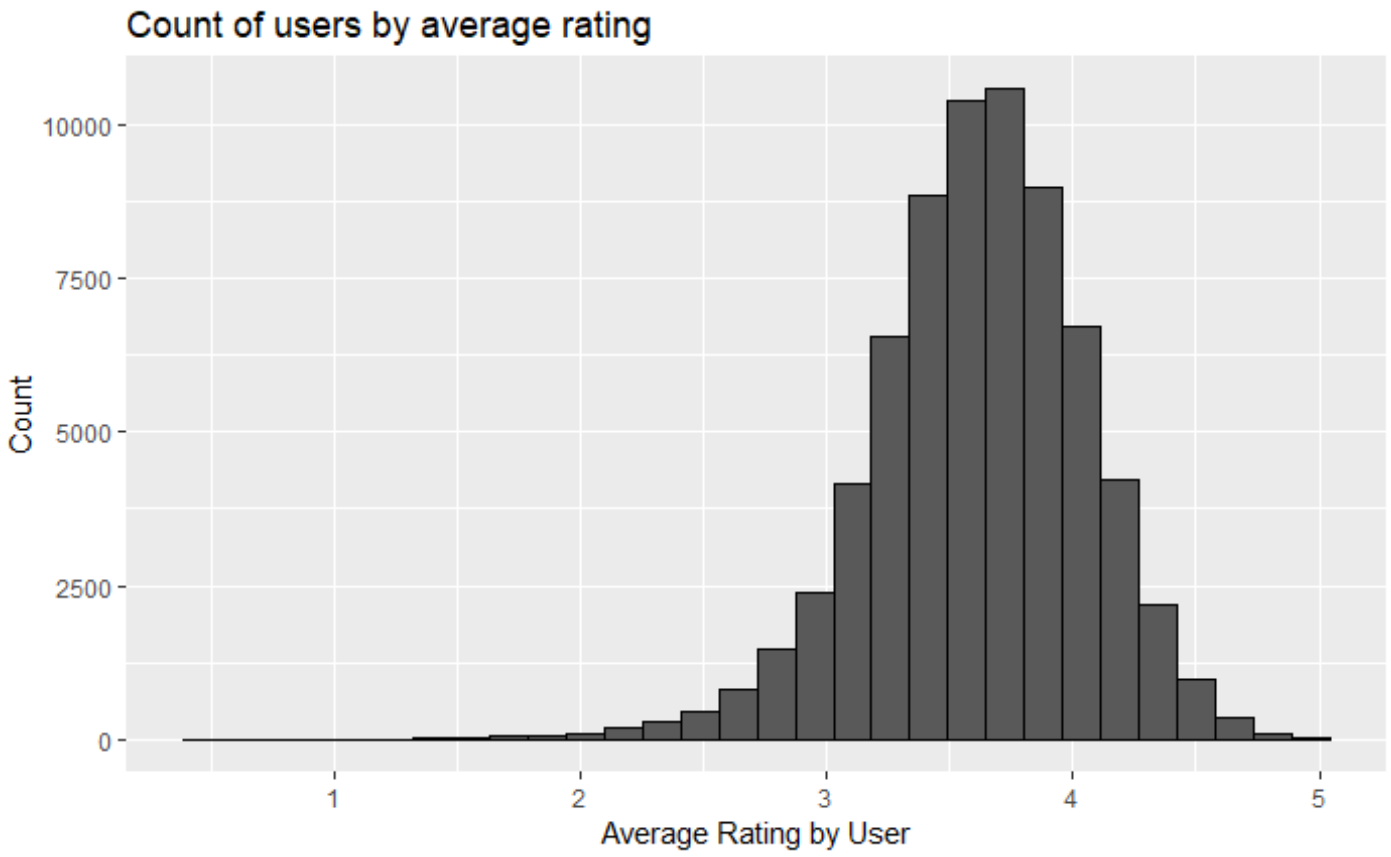
There is a slight improvement.

User Effect

Of course, we can run the same two previous analysis on the userID. Indeed, some users tend to be much more generous than others as we have seen before.

Hide

```
train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")+
  labs(title="Count of users by average rating", x="Average Rating by User", y="Count")
```



Here we can calculate the **user_effect** which is a user-specific effect once the **regularized_movie_effect** has been taken into consideration.

To be clear, this is the model we are trying, where μ is the average rating, $\hat{b}_i(\lambda)$ is the regularized_movie_effect, b_u is the user-specific effect and $\varepsilon_{u,i}$ is the error term:

$$Y_{u,i} = \mu + \hat{b}_i(\lambda) + b_u + \varepsilon_{u,i}$$

Hide

```
user_avgs <- train_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(user_effect = mean(rating - average_rating - reg_movie_effects))
```

Let's create predictions see how it improves our model.

Hide

```
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = average_rating + reg_movie_effects + user_effect) %>%
  .$pred
movieanduser_effect_RMSE <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method= "With Optimized Regularized Movie and User Effect"
,
                                     RMSE=movieanduser_effect_RMSE))
kable(rmse_results,align=rep("c",3), caption = "Metrics calculated on test set only") %>%
  kable_styling(full_width = F) %>%
  column_spec(1,bold=T,border_right = T)
```

Metrics calculated on test set only

Method	RMSE
Netflix Winner	0.8567000
Just the average	1.0600772
With Movie Effect	0.9440170
With Regularized Movie Effect	0.9439867
With Optimized Regularized Movie Effect	0.9439818
With Optimized Regularized Movie and User Effect	0.8660163

Quite the improvement! We are now 24.31% better than we were with just the average.

Perhaps we can get even a little bit better by regularizing the user effect and optimizing its lambda.

Regularizing the User Effect

For the same reasons we regularized the movie effect, we want to also regularize the user effect. In the code below, we find the best λ_u to use for the final model:

$$Y_{u,i} = \mu + \hat{b}_i(\lambda_i) + \hat{b}_u(\lambda_u) + \varepsilon_{u,i}$$

Note that here we are using our test set to perform tuning. This is not best practice, but since the final evaluation will be on the validation set, which we do not use here, we are comfortable doing so.

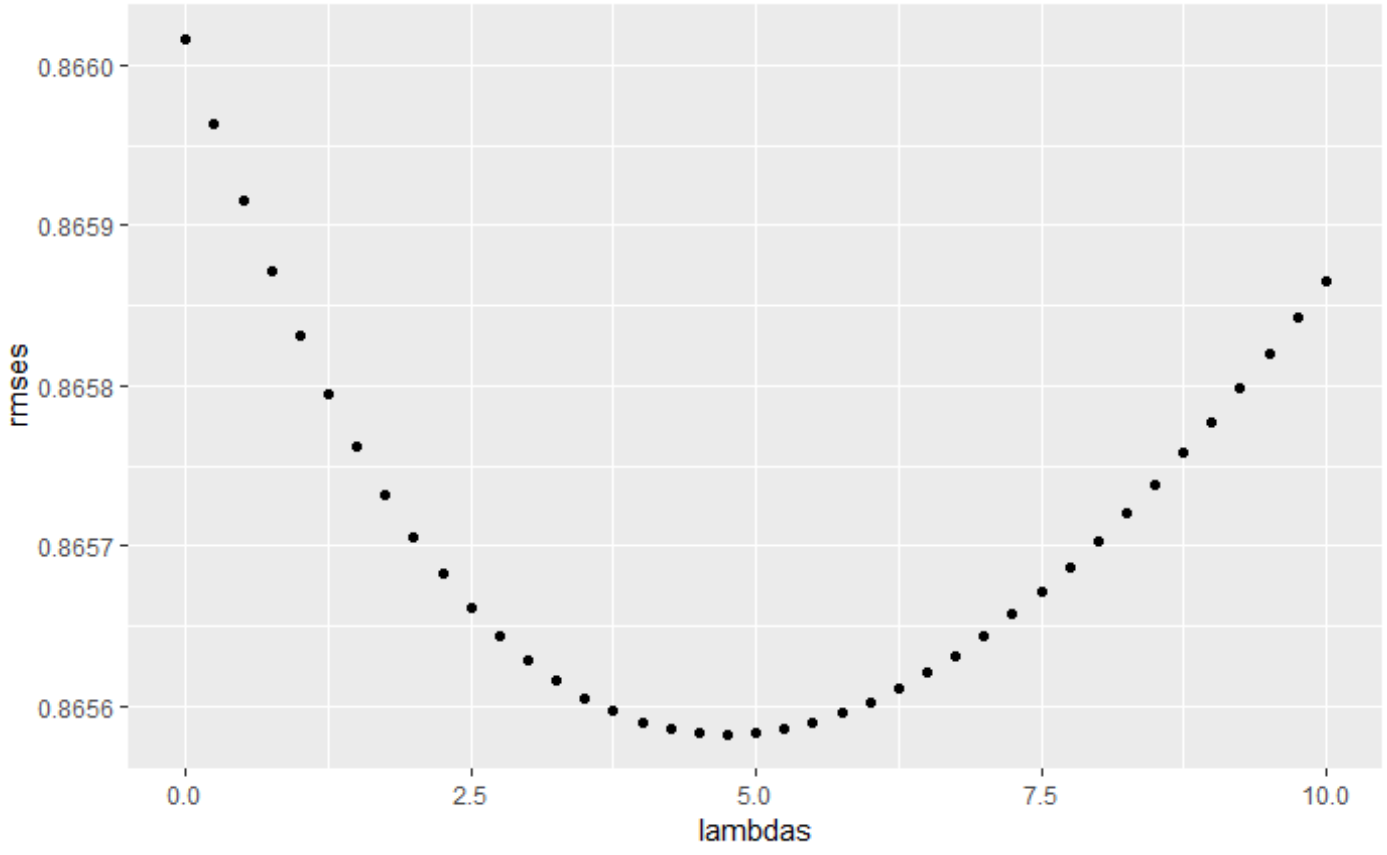
Hide

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)

  reg_b_i <- movie_reg_avgs %>%
    group_by(movieId) %>%
    summarize(reg_b_i = reg_movie_effects)

  reg_b_u <- train_set %>%
    left_join(reg_b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(reg_b_u = sum(rating - reg_b_i - mu)/(n()+l))
  predicted_ratings <-
    test_set %>%
    left_join(reg_b_i, by = "movieId") %>%
    left_join(reg_b_u, by = "userId") %>%
    mutate(pred = mu + reg_b_i + reg_b_u) %>%
    .$pred

    return(RMSE(predicted_ratings, test_set$rating))
})
qplot(lambdas, rmses)
```



Hide

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

Therefore, the best λ_u to use would be 4.75 and the RMSE we expect would be 0.8655826; very good indeed! That is only 1.04% above the Netflix Challenge winners! Obviously, that percent is the hardest to achieve.

Let's update our table.

Hide

```
user_reg_avg <- train_set %>%
    left_join(movie_reg_avgs, by="movieId") %>%
    group_by(userId) %>%
    summarize(reg_user_effects = sum(rating - reg_movie_effects - average_rating)/(n()+lambda_u
ser))
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_reg_avg, by='userId') %>%
  mutate(pred = average_rating + reg_movie_effects + reg_user_effects) %>%
  .$pred
final_RMSE <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method= "With Optimized Regularized Movie and Optimized Re
gularized User Effect",
                                     RMSE=final_RMSE))
kable(rmse_results,align=rep("c",3), caption = "Metrics calculated on test set only") %>%
  kable_styling(full_width = F) %>%
  column_spec(1,bold=T,border_right = T)
```

Metrics calculated on test set only

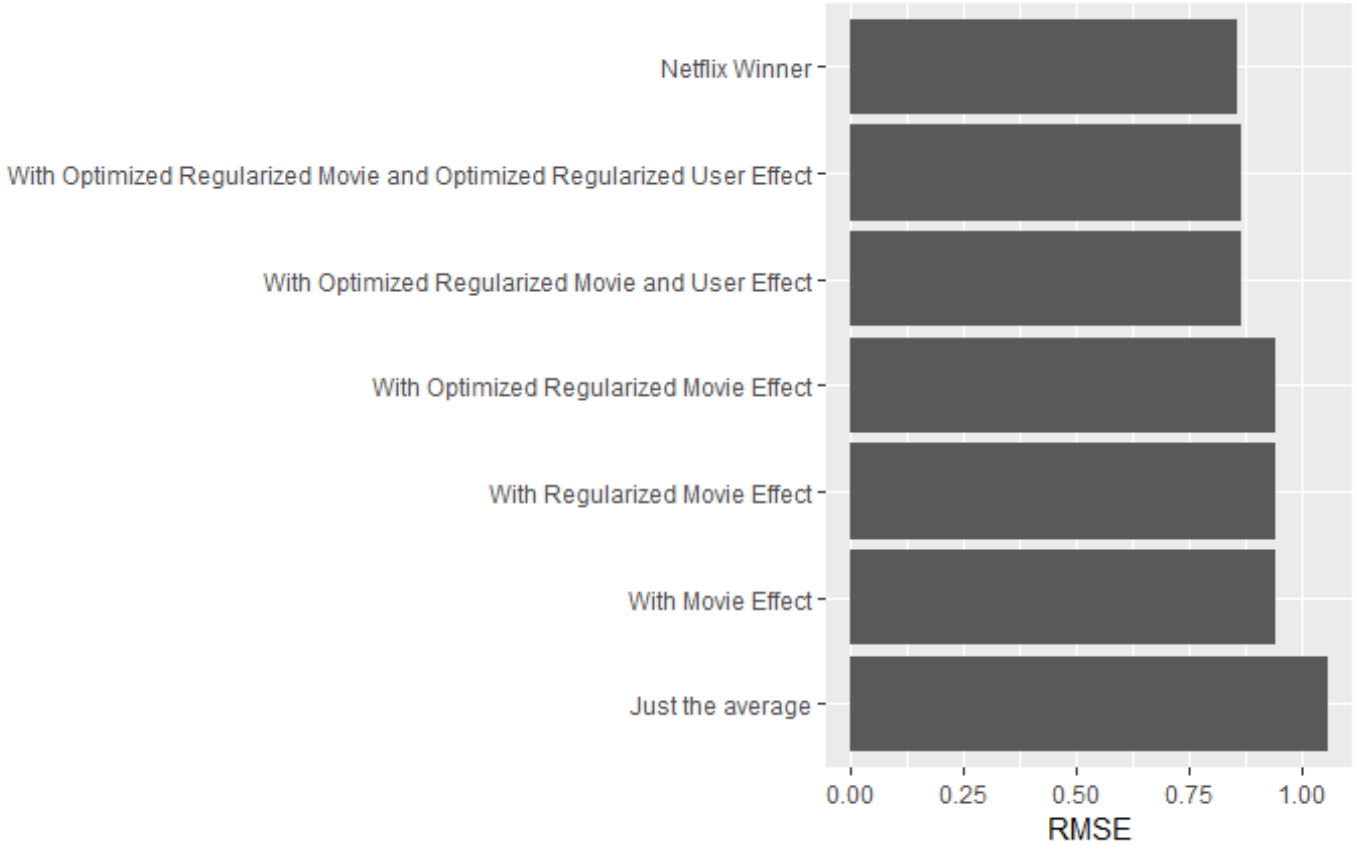
Method	RMSE
Netflix Winner	0.8567000
Just the average	1.0600772
With Movie Effect	0.9440170
With Regularized Movie Effect	0.9439867
With Optimized Regularized Movie Effect	0.9439818
With Optimized Regularized Movie and User Effect	0.8660163
With Optimized Regularized Movie and Optimized Regularized User Effect	0.8655826

Results

Let's now visualize where we are with our RMSEs. Let us not forget that so far we have only been using the **edX** dataset. The ultimate test of our algorithm will be on the **validation** set.

Hide

```
rmse_results %>%
  ggplot(aes(y = RMSE, x = reorder(Method,-RMSE)))+
  geom_bar(stat = "identity")+
  coord_flip()+
  labs(x="",y="RMSE")
```



Let's test this model on the **validation** set!

Hide

```
average_rating <- mean(edx$rating)
movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(reg_movie_effects = sum(rating - average_rating)/(n()+lambda_movie), n_i = n())
reg_b_i <- movie_reg_avgs %>%
    group_by(movieId) %>%
    summarize(reg_b_i = reg_movie_effects)

reg_b_u <- edx %>%
    left_join(reg_b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(reg_b_u = sum(rating - reg_b_i - mu)/(n()+lambda_user))
predicted_ratings <-
    validation %>%
    left_join(reg_b_i, by = "movieId") %>%
    left_join(reg_b_u, by = "userId") %>%
    mutate(pred = mu + reg_b_i + reg_b_u) %>%
    .$pred
```

Let’s verify that we have enough predictions.

- **Rows in Validation**: 999999
- **Predictions**: 999999

Let’s now have a look at our final RMSE

Hide

```
final_RMSE <- RMSE(validation$rating, predicted_ratings)
rmse_results <- bind_rows(rmse_results,
                          data_frame(Method= "Best Model on Validation Set",
                                     RMSE=final_RMSE))
kable(rmse_results,align=rep("c",3)) %>%
  kable_styling(full_width = F) %>%
  column_spec(1,bold=T,border_right = T)
```

Method	RMSE
Netflix Winner	0.8567000
Just the average	1.0600772
With Movie Effect	0.9440170
With Regularized Movie Effect	0.9439867
With Optimized Regularized Movie Effect	0.9439818
With Optimized Regularized Movie and User Effect	0.8660163
With Optimized Regularized Movie and Optimized Regularized User Effect	0.8655826
Best Model on Validation Set	0.8648575

It would seem that, by some luck, our results have improved further. We now boast a RMSE of 0.8648575, which is only 0.95% higher than the Netflix Challenge winners.

Conclusion

In conclusion, we have created a linear model that estimates a movie rating with a RMSE of 0.8648575. This is a very good result, especially when compared to the Netflix Challenge winners and their RMSE of 0.8567.

We did so by using the following model:

$$Y_{u,i} = \mu + \hat{b}_i(\lambda_i) + \hat{b}_u(\lambda_u) + \varepsilon_{u,i}$$

where $\lambda_i = 2$ and $\lambda_u = 4.75$, which are the only two tuning parameters in our model.

To go further and improve our RMSE, we would recommend using the current predicton and using other machine learning techniques, perhaps on some other data like the genre and the timestamp (or perhaps other data available in the full database) to aimed at minimizing the $\varepsilon_{u,i}$. For the purposes of this report, however, we are

satisfied with the final RMSE of 0.8648575, which should award us 25/25 points for this part of the grading.

Note: this report took 11.48 minutes to populate.