Movie Recommendation System

Nathan Siefken

1/23/2019

Introduction:

A recommender system is an information filtering system that tries to predict the ratings a user would give a product. Some of the most dominate companies, like Amazon and Netflix not only have excellent products they provide a personalized experience as an extension of their product. They allow users to rate their products, then they use this information to predict an individual customer's rating for any particular product. They can to a degree of certainty anticipate the needs of their customers.

In the case of Netflix, a low score is a one star and a great movie would be rated five stars. Netflix thought predicting customer ratings was so crucial to their business they hosted a competition in which, they awarded $1,000,000 to anyone who could improve their own recommendation system by 10 percent or more. We will cover one of the methods used by the winners, matrix factorization.

Note: I decided to leave the code in the report in case a reviewer wanted to see it here in addition to my script. In a work setting I would probably leave it out.

```
# This code chunk simply makes sure that all the libraries used here are installed.
packages <- c("knitr","dplyr",  "tidyr", "caret", "ggplot2")
if ( length(missing_pkgs <- setdiff(packages, rownames(installed.packages()))) > 0) {
message("Installing missing package(s): ", paste(missing_pkgs, collapse = ", "))
install.packages(missing_pkgs)
}
```

```
# These are the libraries needed for my report
library(knitr)
library(tidyr)
library(dplyr)
library(ggplot2)
```

The Data :

In this project, I will be creating a movie recommendation system using the 10M MovieLens Dataset. Dataset. This data set includes 10 million ratings on 10 thousand movies by 72 thousand users. 10 % percent of this data was partitioned to create a validation set that will be used as a final test by HarvardX to measure how well our algorithm predicts unseen data. The rest of the data is the data set, edx, used for this project.

I have further partitioned this into my training set, labeled my_train, and a test set, labeled my_test, to test my algorithm before submission.

Methods Section:

Matrix factorization takes information from a sparse matrix like ratings from users on different movies and fills in the empty values with a rating for each movie and each user. The ability to take global input from the data and produce meaningful output is extreamly powerful and we will go through this process in this report.

In comparison, the Neighborhood approach is popular, but it can be very localized. For example if a user likes The Matrix Reloaded, it will suggest they will like The Matrix. It still can be very usefull but at times the results do not seem that insightful.

Measure the quality of the algoriths

Netflix used the used Root Mean Squared Errors (RMSE) to measure the quality of the algorithms in the challenge. The winning team Bellcore Pragmatic Chaos won the challenge with a RMSE of .8558 which was an improvement of 10.05 of Netflix's recommender system. The goal of this report is to beat the results of Bellcore Pragmatic Chaos and have a lower RMSE than .85558.

We define $y_{u,i}$ as the rating for movie $\imath$ and user u and denote our prediction as $\hat{y}_{i,u}$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{d_i - f_i}{\sigma_i}\right)^2}$$

As described in Wikipedia

> Root Mean Square Error (RMSE) is a frequently used measure of the differences between values (sample or population values) predicted by a model or an estimator and the values observed.

Since it is a measurement of error, the smaller the RMSE the better. To better understand how to interpret the RMSE for our algorithm, if we receive a RMSE of 1, means that our prediction was off by an average of 1 star.

We should start by building a function that will calculate the RMSE for actual values (true_ratings) from our test set to their corresponding predictors from our models:

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

Exploration of the data:

```
dim(my_train)
```

```
## [1] 7200043       6
```

The training set has 7,200,043 ratings with the following 6 variables.

```
colnames(my_train)
```

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

This code will show the number of unique movies and users.

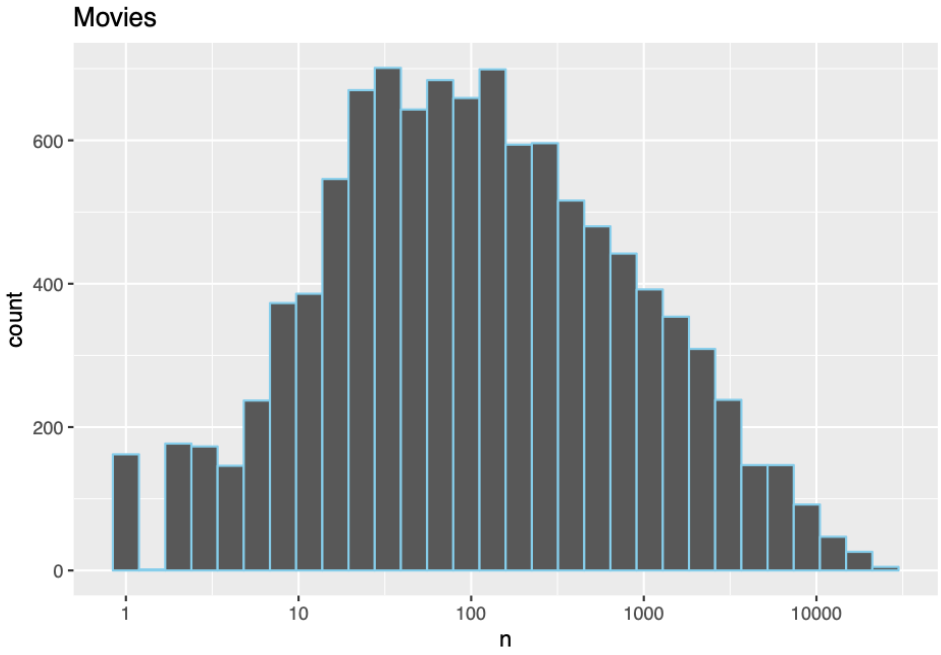
```
my_train %>%
  summarize(Users= n_distinct(userId),
            Movies = n_distinct(movieId) )
```

```
##   Users Movies
## 1 69878  10641
```

My training data set has 69,878 users have rated 10,641 different movies for a grand total of 7,200,043 observations. For each of the observations we have the following variables: userId, movieId, rating, timestamp, title, and genres.

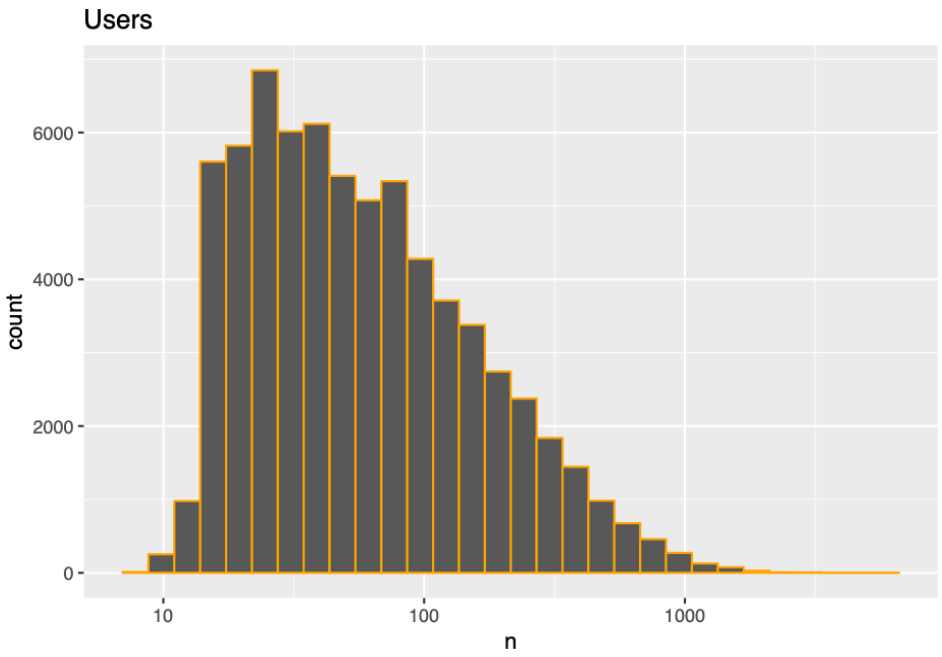
Below shows a histogram the number of times a few movies have been rated. The movies that have been rated more than 10,000 times is going to carry much more weight in a calculation than the movies on the left side of the graph that have been rated only 10 times or fewer. This is known as the movie bias or effect.

```
my_train %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "skyblue") +
  scale_x_log10() +
  ggtitle("Movies")
```



We can see there appears to be a similar bias among users. Some users are extremely active while some have only reviewed a few movies.

```
my_train %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "orange") +
  scale_x_log10() +
  ggtitle("Users")
```



<p>

To get a better perspective of our data, let's see the top 10 genres for the movies in our data set.

```
genreCount <-my_train %>% separate_rows(genres, sep = "\\|") %>%
group_by(genres) %>%
summarize(count = n()) %>%
arrange(desc(count)) %>%
head(n=10)
genreCount %>% knitr::kable()
```

genres	count
Drama	3129429
Comedy	2832418
Action	2048241
Thriller	1860181
Adventure	1526542
Romance	1370069
Sci-Fi	1072813
Crime	1062327
Fantasy	739821
Children	590222

Drama and Comedy are the most likely to be rated.

The following are the top 10 most rated the movies.

```
Top10 <- my_train %>% group_by(movieId, title) %>%
summarize(count = n()) %>%
arrange(desc(count)) %>% head(n=10)
Top10 %>% knitr::kable()
```

movieId	title	count
296	Pulp Fiction (1994)	25100
356	Forrest Gump (1994)	24936
593	Silence of the Lambs, The (1991)	24268
480	Jurassic Park (1993)	23487
318	Shawshank Redemption, The (1994)	22376
110	Braveheart (1995)	21012
457	Fugitive, The (1993)	20827
589	Terminator 2: Judgment Day (1991)	20807
260	Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	20582
150	Apollo 13 (1995)	19483

Let's explore to see the most common rating given by the users.

```
ratings <- my_train %>% group_by(rating) %>% summarize(count = n()) %>%
  mutate(percent = count/nrow(my_train)*100) %>%
  top_n(5) %>%
  arrange(desc(count))
```

```
## Selecting by percent
```

```
ratings %>% knitr::kable()
```

rating	count	percent
4.0	2070621	28.758453
3.0	1696777	23.566206
5.0	1111767	15.441116
3.5	633422	8.797475
2.0	569256	7.906286

It is important to note that the users rated movies 4 stars over a quarter of the time.

Now that we have taken a look at the data, we should start building our recommendation machine

Baseline

We will start with our baseline, the most basic recommendation system possible. This is the average for all users across all movies and use this average to predict our ratings.

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

```
mu_hat <- mean(my_train$rating)

mu_hat
```

```
## [1] 3.512351
```

Now that we have our $\hat{\mu}$ we can determine RMSE for our baseline method.

```
baseline_rmse <- RMSE(my_test$rating, mu_hat)

baseline_rmse
```

```
## [1] 1.059951
```

We are getting a RMSE of about 1.06. Our prediction are on average one star off the actual data. Not all that accurate. However, for this model, this is the lowest RMSE we can have. To demonstrate we will use 4, the most common rating given by the users. We can see below that our RMSE is higher than when we use μ.

```
predictions <- rep(4, nrow(my_test))
RMSE(my_test$rating, predictions)
```

```
## [1] 1.166504
```

The following code will keep track with all of our test with this matrix.

```
rmse_results <- data_frame(method = "Baseline", RMSE = baseline_rmse)

rmse_results %>% knitr::kable()
```

method	RMSE
Baseline	1.059951

This model utilizes μ but does not take into account the movie or user bias, so there is opportunity for improvement.

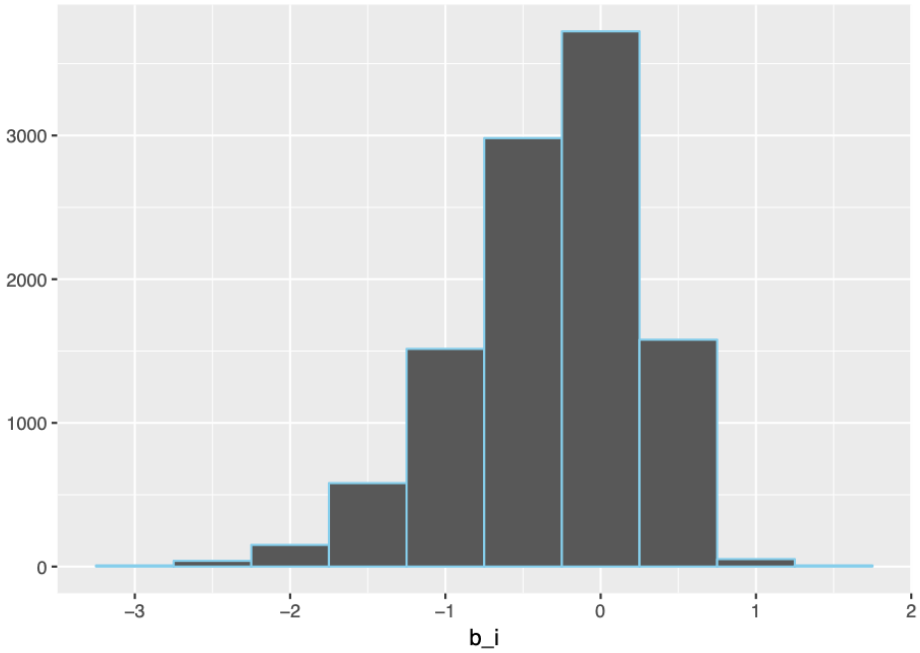
Modeling Movie Effect

To improve on our model, we should consider the movie bias i, as we mentioned the uneven amount of times movies are rated. The histogram below support this claim with data.

Just to better explain the graph. μ in our case is 3.5, therefore a b_i of -3 is our lowest rating of .5 and b_i of 1.5 would b a five-star rating.

```
mu <- mean(my_train$rating)
movie_avgs <- my_train %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

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



Note: The lm function is excellent to use but we have too many values in this data set and it will take too long to run, possibly error out due to lack of computer memory.

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

In this situation we can estimate the movie bias $\hat{b}_i$ is just the average of $y_{u,i} - \mu$ for each movie $\imath$. So we can compute them his way (for computational ease in r we will drop the hat notation)

Let's see how our model did with the following equation. We will name it the "Movie Effect Model"

$$\hat{y}_{u,i} = \hat{\mu} + \hat{b}_i$$

```
predicted_ratings <- mu_hat + my_test %>%
  left_join(movie_avgs, by='movieId') %>%
```

```
  .$b_i

model_1_rmse <- RMSE(predicted_ratings, my_test$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_1_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Baseline	1.059951
Movie Effect Model	0.943450

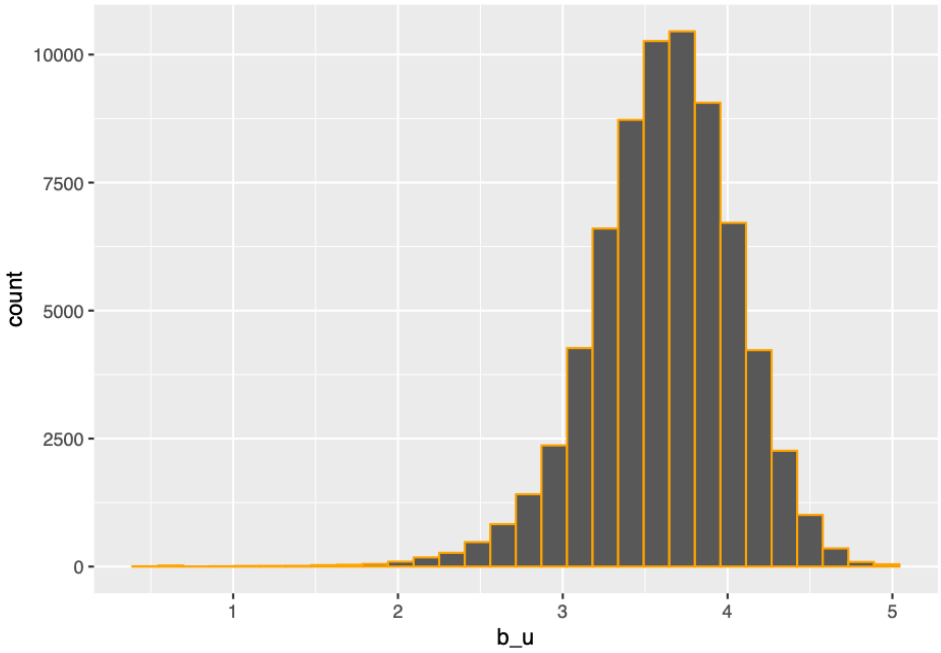
There is a modest improvement when we include the movie bias to our baseline model.

Movie and User Effects Model

There is further opportunity to improve our model by including the users bias also known as the user effect b_u.

We can visualize in the difference in the nature of the users rating habits in the histogram below.

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



As you can see the users on the left on average rate movies poorly and the users on the right on average enjoy every movie. Taking this into account should improve our model.

```
user_avgs <- my_train %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))
```

Below we can see how our base model has improved when we include both the movie and user bias to our baseline.

```
predicted_ratings <- my_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
```

```
  .$pred


model_2_rmse <- RMSE(predicted_ratings, my_test$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = model_2_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Baseline	1.0599514
Movie Effect Model	0.9434500
Movie + User Effects Model	0.8662287

This is a great improvement from our baseline and is clse to what the winners of the Netflix achieved. I think it is worth taking a look at our results to see if there is an additional opportunity for improvement.

Regularization

First, let's look at our largest errors in our *Movie Effect Model* to see if we can understand what happened. Here is a list of the largest errors.

```
my_test %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(residual = rating - (mu_hat + b_i)) %>%
  arrange(desc(abs(residual))) %>%
  select(title,  residual) %>% slice(1:10) %>% knitr::kable()
```

title	residual
Shawshank Redemption, The (1994)	-3.954125
Shawshank Redemption, The (1994)	-3.954125
Godfather, The (1972)	-3.915781
Godfather, The (1972)	-3.915781
Godfather, The (1972)	-3.915781
Godfather, The (1972)	-3.915781
Godfather, The (1972)	-3.915781
Godfather, The (1972)	-3.915781
Godfather, The (1972)	-3.915781
Usual Suspects, The (1995)	-3.866495

The first thing I notice is that these movies repeat, so the error repeats many times in our matrix. We will run the following code that will eliminate duplicated movies errors to better understand unique mistakes.

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

The following code shows will produce the 10 best movies according to our model.

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

title	b_i
Hellhounds on My Trail (1999)	1.487649
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.487649
Satan's Tango (Sátántangó) (1994)	1.487649
Appaloosa, The (1966)	1.487649
Shadows of Forgotten Ancestors (1964)	1.487649
Fighting Elegy (Kenka erejii) (1966)	1.487649
Sun Alley (Sonnenallee) (1999)	1.487649
Blue Light, The (Das Blaue Licht) (1932)	1.487649
Human Condition III, The (Ningen no joken III) (1961)	1.237649
Constantine's Sword (2007)	1.237649

This is interesting that all of these movies are unrecognizable. It doesn't seem fitting for the best movies rated by users.

These are the 10 worst movies according to our model.

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

title	b_i
State Property (2002)	-3.012351
Besotted (2001)	-3.012351
Hi-Line, The (1999)	-3.012351
Accused (Anklaget) (2005)	-3.012351
Confessions of a Superhero (2007)	-3.012351
War of the Worlds 2: The Next Wave (2008)	-3.012351
Hip Hop Witch, Da (2000)	-2.901240
SuperBabies: Baby Geniuses 2 (2004)	-2.767907
Disaster Movie (2008)	-2.692351
From Justin to Kelly (2003)	-2.592598

Now I would be curious to know how many times these movies have been rated.

The following counts how many times our top 10 movies have been rated in our training set.

```
my_train %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

```
## Joining, by = "movieId"
```

title	b_i	n
Hellhounds on My Trail (1999)	1.487649	1
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.487649	1
Satan's Tango (Sátántangó) (1994)	1.487649	2
Appaloosa, The (1966)	1.487649	1
Shadows of Forgotten Ancestors (1964)	1.487649	1
Fighting Elegy (Kenka erejii) (1966)	1.487649	1
Sun Alley (Sonnenallee) (1999)	1.487649	1
Blue Light, The (Das Blaue Licht) (1932)	1.487649	1
Human Condition III, The (Ningen no joken III) (1961)	1.237649	4
Constantine's Sword (2007)	1.237649	2

The following counts how many times our top 10 worst movies have been rated in our training set.

```
my_train %>% count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i, n) %>%
```

```
  slice(1:10) %>%
  knitr::kable()
```

```
## Joining, by = "movieId"
```

title	b_i	n
State Property (2002)	-3.012351	1
Besotted (2001)	-3.012351	2
Hi-Line, The (1999)	-3.012351	1
Accused (Anklaget) (2005)	-3.012351	1
Confessions of a Superhero (2007)	-3.012351	1
War of the Worlds 2: The Next Wave (2008)	-3.012351	1
Hip Hop Witch, Da (2000)	-2.901240	9
SuperBabies: Baby Geniuses 2 (2004)	-2.767907	45
Disaster Movie (2008)	-2.692351	25
From Justin to Kelly (2003)	-2.592598	162

Just as we suspected these movies that raked at the top and bottom were only rated a few times. These movies do not have enough ratings to reflect a true rating of the movie or perhaps they do reflect the true rating. The point is we do not know, and we should be skeptical of these results.

To optimize b_i the following equation can be used. This is the least squares and the second term is the penalty which gets lareger as b_i gets larger.

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

This is equation can be reduced to the following equation.

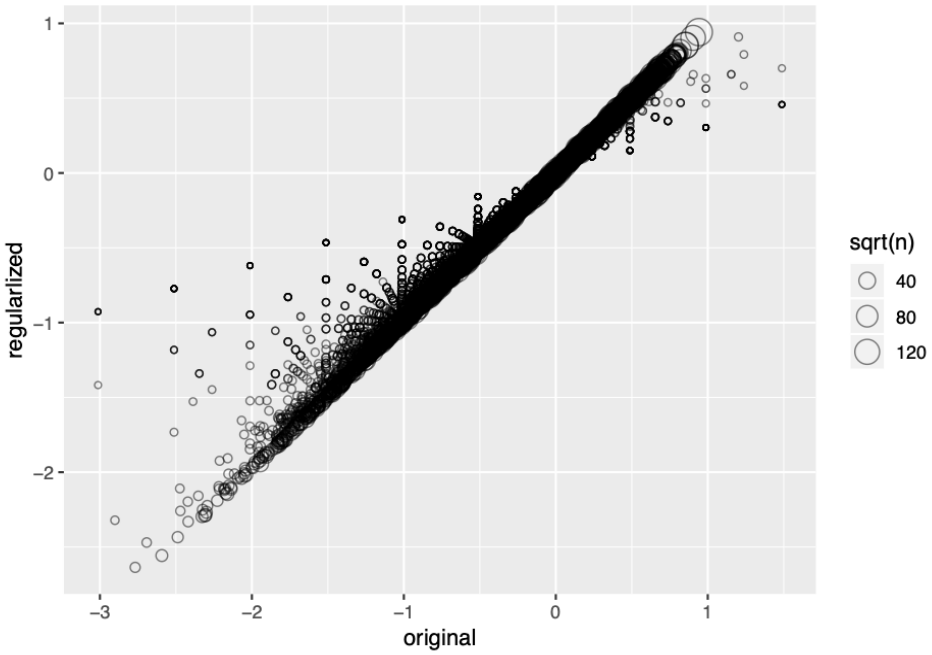
$$\hat{b}_i(\lambda)=\frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}(Y_{u,i}-\hat{\mu})$$

We can use the method Regularization which adds a penalty lambda (in this case 2.25) to penalizes movies with large estimates from a small sample size, in our case only rated a few times.

```
lambda <- 2.25
mu <- mean(my_train$rating)
movie_reg_avgs <- my_train %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())
```

The next code creates a graph that shows how the estimates shrink with the penalty.

```
data_frame(original = movie_avgs$b_i,
           regularlized = movie_reg_avgs$b_i,
           n = movie_reg_avgs$n_i) %>%
  ggplot(aes(original, regularlized, size=sqrt(n))) +
  geom_point(shape=1, alpha=0.5)
```


regularlized
original
sqrt(n)
40
80
120

These are the 10 best rated movies after regularization. This makes a lot more sense because these are recognizable and have been rated thousands of times.

```
my_train %>%
  count(movieId) %>%
  left_join(movie_reg_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

```
## Joining, by = "movieId"
```

title	b_i	n
Shawshank Redemption, The (1994)	0.9416792	22376
More (1998)	0.9095722	7
Godfather, The (1972)	0.9032873	14213
Usual Suspects, The (1995)	0.8540323	17254
Schindler's List (1993)	0.8511192	18640
Casablanca (1942)	0.8159745	8920
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	0.8049260	2303
Rear Window (1954)	0.8007511	6350
Double Indemnity (1944)	0.7970726	1741
Third Man, The (1949)	0.7955756	2373

These are the 10 worst rated movies after regularization. Even though they may not be recognizable, we can trust their rating as they have been rated several times.

```
my_train %>%
  count(movieId) %>%
  left_join(movie_reg_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

```
## Joining, by = "movieId"
```

title	b_i	n
SuperBabies: Baby Geniuses 2 (2004)	-2.636102	45
From Justin to Kelly (2003)	-2.557083	162
Disaster Movie (2008)	-2.470047	25
Pokémon Heroes (2003)	-2.434147	102
Carnosaur 3: Primal Species (1996)	-2.330265	59
Hip Hop Witch, Da (2000)	-2.320992	9
Barney's Great Adventure (1998)	-2.296600	163
Glitter (2001)	-2.289757	259
Gigli (2003)	-2.285400	265
Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002)	-2.267890	162

This code measures how well our predictions preformed on the my_test set and then adds the RMSE to our

results table.

```
predicted_ratings <- my_test%>%
  left_join(movie_reg_avgs, by='movieId') %>%
  mutate(pred = mu_hat + b_i) %>%
  .$pred

model_3_rmse <- RMSE(predicted_ratings, my_test$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie Effect Model",
                                     RMSE = model_3_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Baseline	1.0599514
Movie Effect Model	0.9434500
Movie + User Effects Model	0.8662287
Regularized Movie Effect Model	0.9433732

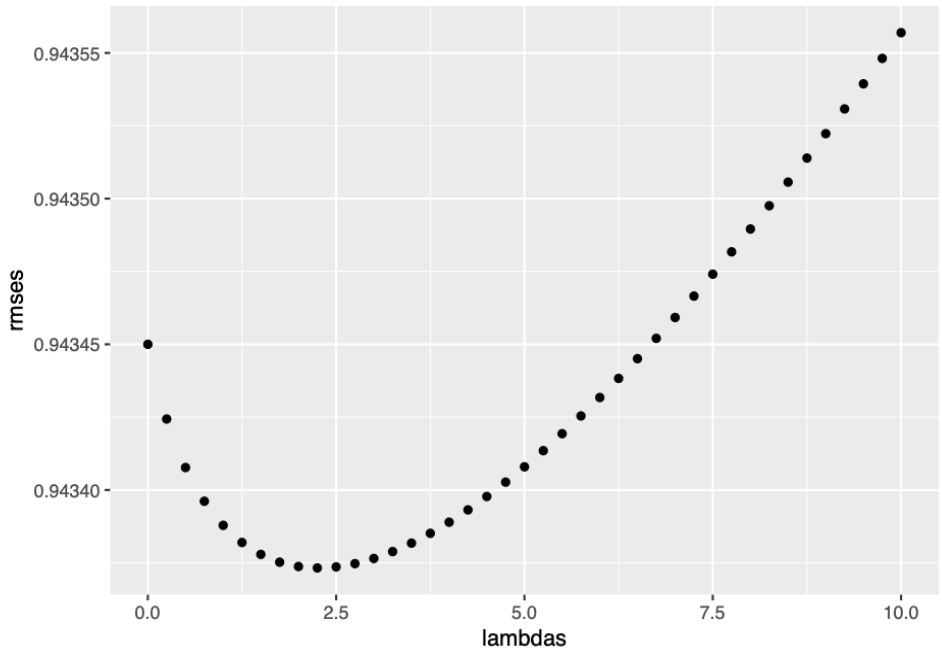
The result is slightly better than the results from the Movie Effect Model without Regularization. However now when you look at the data it makes more sense. The best movies are blockbusters and the worse movies are movies that have at least been rated enough for us to trust the rating.

Picking a penalty

The following code shows how use cross validation to choose lambda. It will essentially find the RMSE for different penalties (from 0 to 10 at a sequence of .25) applied to the movie bias. We then can easily find the lambda that will produce the lowest RMSE.

We then can visualize the results of the different lambdas on a graph and we print out the λ that produces the smallest RMSE for b_i.

```
lambdas <- seq(0, 10, 0.25)

mu <- mean(my_train$rating)
just_the_sum <- my_train %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())

rmses <- sapply(lambdas, function(l){
  predicted_ratings <- my_test %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    .$pred
  return(RMSE(predicted_ratings, my_test$rating))
})
qplot(lambdas, rmses)
```




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

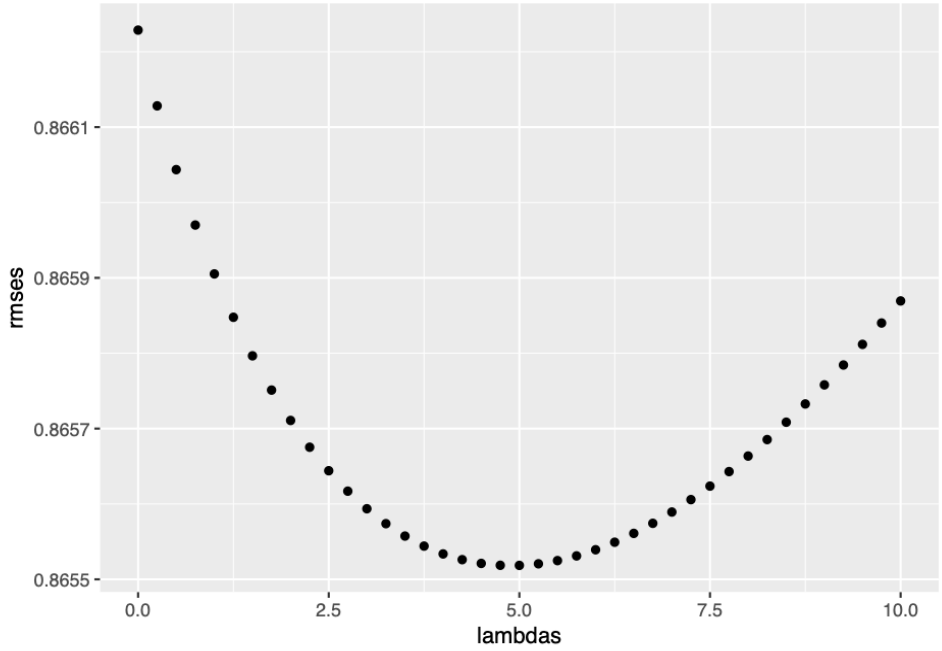
```
## [1] 2.25
```

We will repeat the step above and apply it to the user and movie bias to find the λ that gives us the smallest RMSE.

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

Again, we can visualize the results of the different lambdas on a graph and then print out the λ that produces the smallest RMSE for b_i.

```
lambdas <- seq(0, 10, 0.25)

rmses <- sapply(lambdas, function(l){

  mu <- mean(my_train$rating)

  b_i <- my_train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  b_u <- my_train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  predicted_ratings <-
    my_test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    .$pred

  return(RMSE(predicted_ratings, my_test$rating))
})

qplot(lambdas, rmses)
```



This following is the λ for the lowest RMSE

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

```
## [1] 5
```

Regularization on the Movie + User Effect Model had the best results.

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User Effect Model",
                                     RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Baseline	1.0599514
Movie Effect Model	0.9434500
Movie + User Effects Model	0.8662287
Regularized Movie Effect Model	0.9433732
Regularized Movie + User Effect Model	0.8655185

Conclusion:

We have effectivly used machine learning methods to create a movie recommendation system. The methods that we used did not quite beat the winners of the $1 million Netfilx challange but we did get close. We found that when we take into account both the user and the movie effect or bias we notice the largest improvement in RMSE. Also when we apply Regularization we do see a slight improvement on the Movie Effect Model and User + Movie Effect Model. However, the improvement is espcially effective on movies with small sample sizes and allows us to produce results that make sense.