MovieLens Recommendation Model

Daniel Romotsky

10/15/2019

1. The Project

The objective of this project is to build a model that predicts a rating for a given movie for a given user. I am using a public data set from grouplens.org, which includes 10 million ratings of over 10 thousand movies from over 72 thousand users.

The key steps I've taken in this project is to:
- Download the full data set.
- Data cleansing into one table.
- Split table into a training (90%) and test (10%) set.
- Apply data wrangling techniques to add new variables.
- Preprocessing and data visualization.
- Test different modeling techniques, using the Root Mean Square Error as the measure of success.

Root Mean Square Error (AKA RMSE) will be the measure of accuracy of the model.

The calculation used here is $\sqrt(1/n\sum_1^n(trueratings_n - predictedratings_n)^2)$

Download the data set

First, I need to download the data set from the grouplens site… *Please refer to the R script for these steps.*

A view into the original data set:

```
## Observations: 10,000,054
## Variables: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ movieId   <int> 122, 185, 231, 292, 316, 329, 355, 356, 362, 364, 37...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5...
## $ timestamp <int> 838985046, 838983525, 838983392, 838983421, 83898339...
## $ title     <fct> "Boomerang (1992)", "Net, The (1995)", "Dumb & Dumbe...
## $ genres    <fct> Comedy|Romance, Action|Crime|Thriller, Comedy, Actio...
```

unique_mov	unique_users	unique_genres
10677	69878	797

Create Train vs Test Groups

Next, we split the data set into a training (90%) and test (10%) set using the caret package:

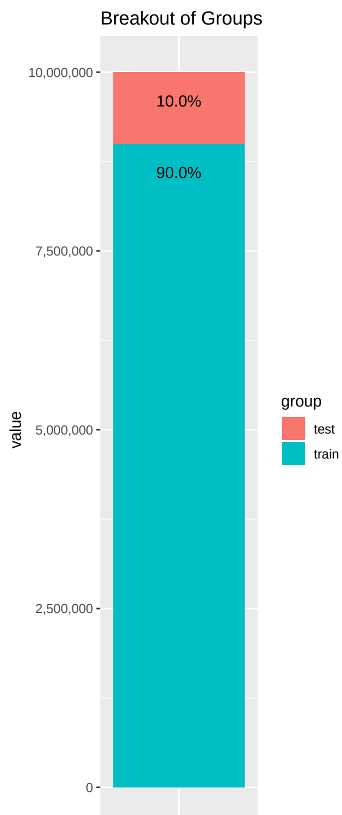
Training set

```
## Observations: 9,000,055
## Variables: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1...
## $ movieId   <int> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 37...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5...
```

```
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 83898339...
## $ title     <fct> "Boomerang (1992)", "Net, The (1995)", "Outbreak (19...
## $ genres    <fct> Comedy|Romance, Action|Crime|Thriller, Action|Drama|...
```

Test set

```
## Observations: 999,999
## Variables: 6
## $ userId    <int> 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5...
## $ movieId   <int> 231, 480, 586, 151, 858, 1544, 590, 4995, 34, 432, 4...
## $ rating    <dbl> 5.0, 5.0, 5.0, 3.0, 2.0, 3.0, 3.5, 4.5, 5.0, 3.0, 3....
## $ timestamp <int> 838983392, 838983653, 838984068, 868246450, 86824564...
## $ title     <fct> "Dumb & Dumber (1994)", "Jurassic Park (1993)", "Hom...
## $ genres    <fct> Comedy, Action|Adventure|Sci-Fi|Thriller, Children|C...
```



Now we're ready to start building the model. I will be using solely the *edx* data set for training purposes and testing my model on the *validation* set.

2. *Methods & Analysis*

My approach to finding an acceptable final model was to increment different effects based on the variables provided. The first approach was to use the overall average for all the observations in the training set.

Overall Average

mu <- edx %>% summarise(mean(rating)) %>% pull() # overall overage rating Which is:

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

We then use mu as the predictor on the validation set and compare it to the actual rankings

```
## predict ratings using mu
predict_mu <- validation %>% mutate(pred = mu) %>% select(rating, pred)
predict_mu %>% summarise(RMSE(rating, pred)) ## over a whole rating off on average... not good
```

```
##   RMSE(rating, pred)
## 1           1.061202
```

Over 1 full star rating off is not great so overall average is definitely not good enough.

Movie Effect

I can assume that the movie itself is a strong predictor for rating and can calculate the effects by taking the mean of the difference in movie rank and overall average rank with:

```
movie_avg <- edx %>% group_by(movieId) %>% summarise(bm = mean(rating - mu))

glimpse(movie_avg)
```

```
## Observations: 10,677
## Variables: 2
## $ movieId <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,...
## $ bm      <dbl> 0.41517246, -0.30706581, -0.36548171, -0.64816590, -0....
```

Now that we have movie effects for the unique 10.6k movies, we can use that to estimate ratings against the test set. We join this back into the validation set and add it to the baseline mu to get our latest predictions. We then compare it against the true ratings to calculate RMSE.

```
## join that back into the test data to predict ratings with movie bias
predict_mu_bm <- validation %>% left_join(movie_avg) %>% mutate(mu = mu, pred = mu + bm) %>% pull(pred)
RMSE(validation$rating, predict_mu_bm)
```

```
## [1] 0.9439087
```

That made a significant difference

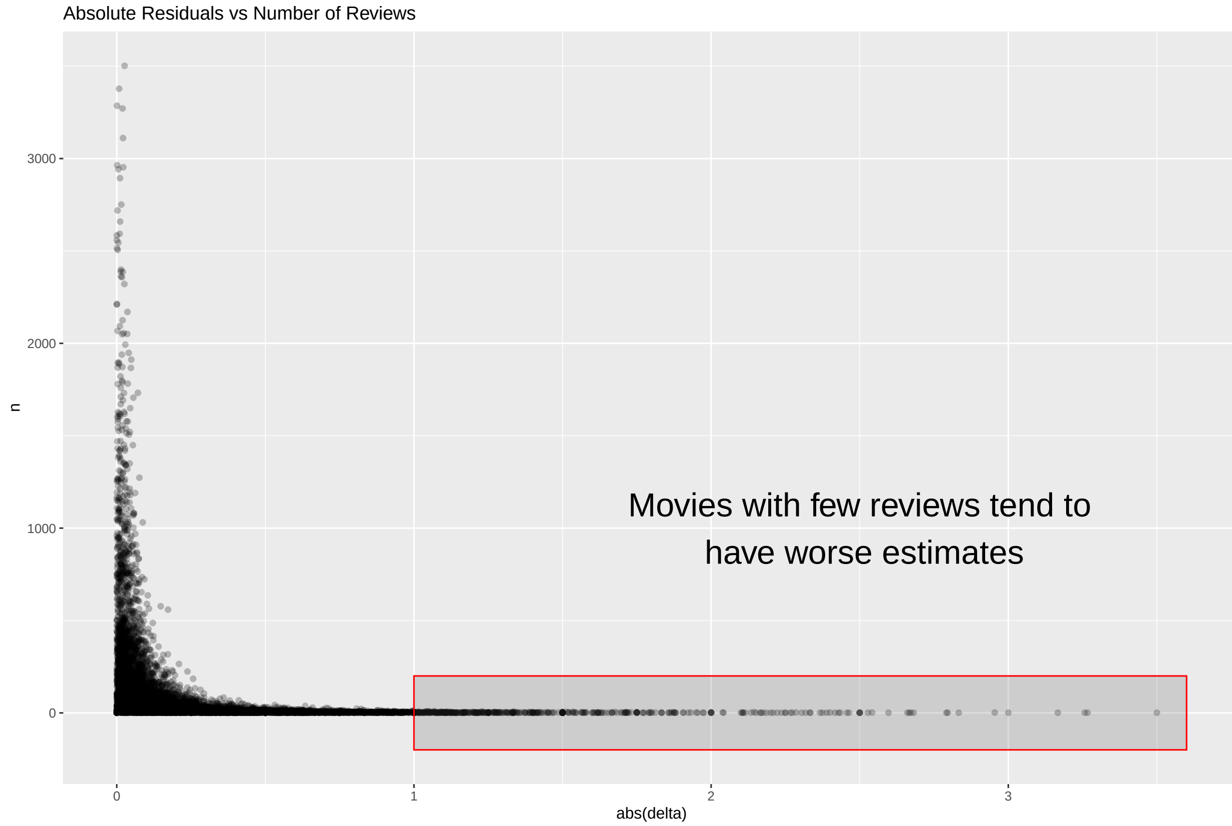
```
## RMSE difference between Naive prediction and adding movie effect is -0.1172931
```

But can we do better? Which movies have the worst predictions?

movieId	title	avg_r	avg_p	delta	n
31692	Uncle Nino (2003)	0.5	4.00	-3.50	1
25945	They Live by Night (1948)	0.5	3.77	-3.27	1
7785	Time For Drunken Horses, A (Zamani barayé masti asbha) (2000)	0.5	3.76	-3.26	1
56030	Darfur Now (2007)	0.5	3.67	-3.17	1
3220	Night Tide (1961)	0.5	3.50	-3.00	1
30783	Blood and Black Lace (Sei donne per l'assassino) (1964)	0.5	3.45	-2.95	2
4070	Amy (1998)	0.5	3.33	-2.83	1
7253	It (1927)	0.5	3.30	-2.80	1
6556	Sea Is Watching, The (Umi wa miteita) (2002)	0.5	3.29	-2.79	1
6138	Tag: The Assassination Game (a.k.a. Everybody Gets It in the End) (1982)	0.5	3.18	-2.68	1

Looks like most of these have 1-2 reviews which makes it tougher to predict.

Plotting out the distribution of predictions, you can see that many have few reviews.



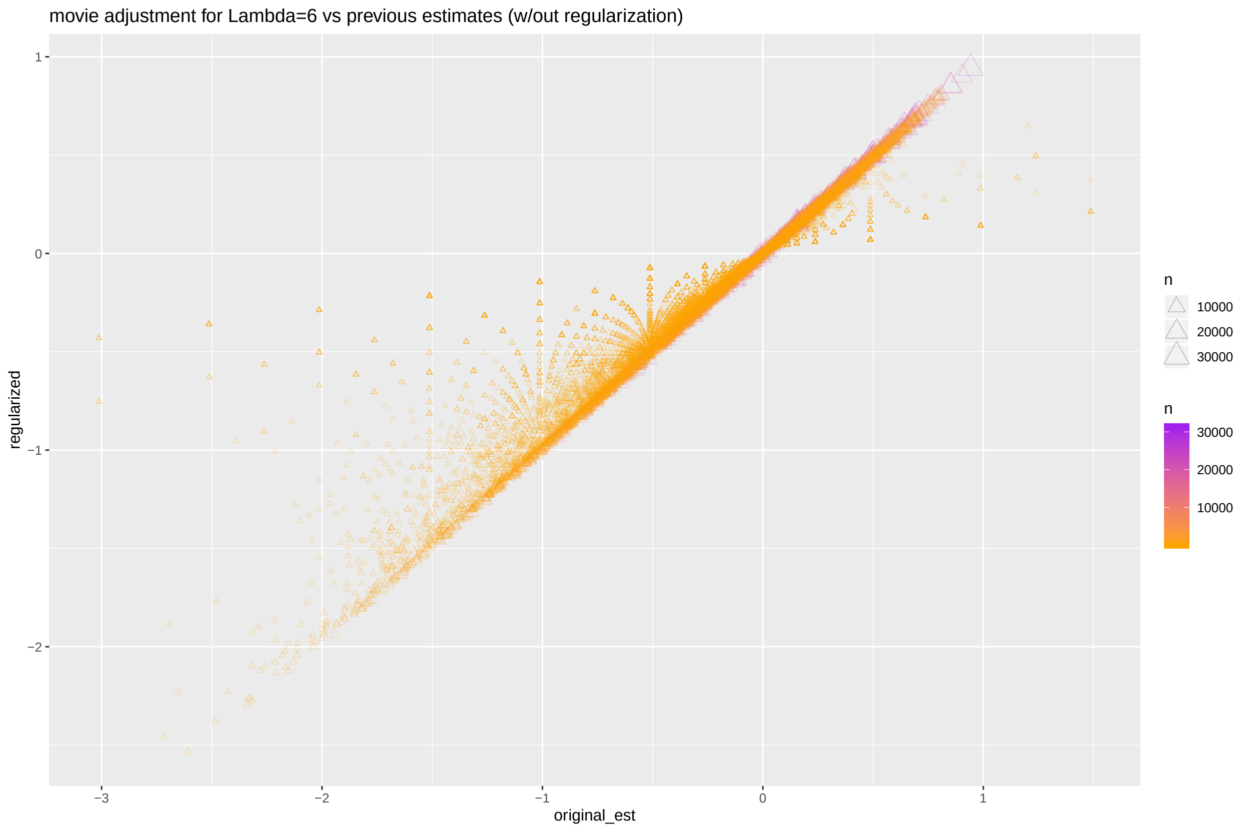
Regularization

For those movies with very few reviews, I want to shrink estimates. Starting with an arbitrary lambda, let's inspect how we can regularize our estimates. Instead of taking the standard difference or avg vs rating, we include lambda and number of reviews to normalize the data.

```
lambda <- 6 # arbitrary lambda
b_m = sum(rating - mu)/(n()+lambda) ## revised movie effect
```

If n = number of reviews is large, lamba will not change the results much. If n is small, lamba will bring the movie effect closer to zero.

How is this different from the standard average?



Many of the movies with small n (aka reviews) move towards zero under regularization.

Now let's look at the worst rated movies with regularization implemented:

movieId	title	b_m	n
6483	From Justin to Kelly (2003)	-2.53	199
8859	SuperBabies: Baby Geniuses 2 (2004)	-2.45	56
6371	Pokémon Heroes (2003)	-2.38	137
4775	Glitter (2001)	-2.30	339
6587	Gigli (2003)	-2.28	313
5672	Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002)	-2.27	202
1826	Barney's Great Adventure (1998)	-2.26	208
61348	Disaster Movie (2008)	-2.23	32
3574	Carnosaur 3: Primal Species (1996)	-2.23	68
31698	Son of the Mask (2005)	-2.13	165

Let's apply this lambda to our predictions:

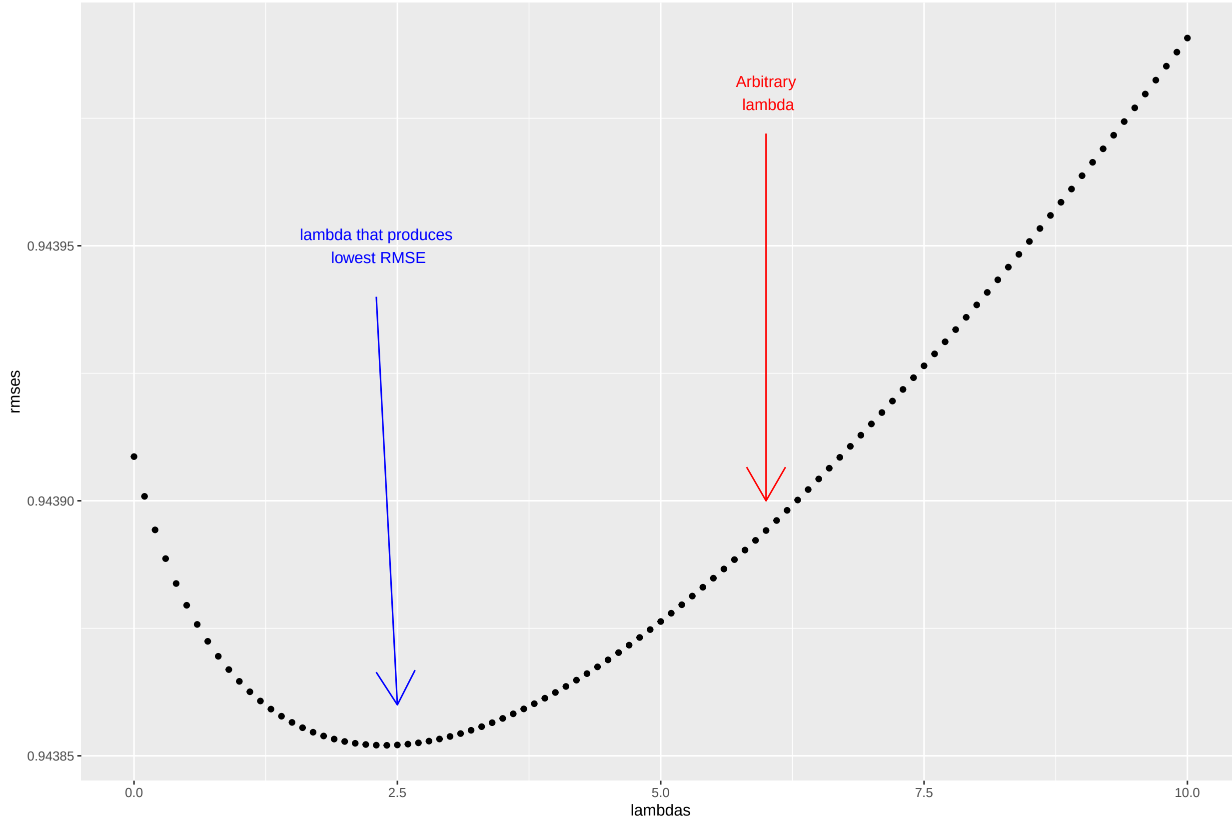
```
predict_mu_bm_reg <- mu + validation %>% left_join(movie_reg_avgs, by="movieId") %>% pull(b_m)
cat("RMSE with movie effect lambda=6 is",RMSE(validation$rating, predict_mu_bm_reg))
```

```
## RMSE with movie effect lambda=6 is 0.9438942
```

That's not much better than our unregularized prediction

```
## Improvement of adding regularization with lambda=6 is -1.448894e-05
```

This is because we picked an *arbitrary lambda*. Let's create a function that tries different lambdas to find the lambda that yields the best RMSE against the test set. See R code for function




```
## Lowest Lambda from function is 2.4
```

Now we can use the lambda with the best impact to RMSE and run predictions again.

```r
lambda <- lambdas[which.min(rmses)]
movie_reg_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(bi = sum(rating - mu)/(n()+lambda), n_i = n())

##now let's see if it improved accuracy
predict_mu_bi_reg <- mu + validation %>% left_join(movie_reg_avgs) %>% pull(bi)
RMSE(validation$rating, predict_mu_bi_reg)
```

```
## [1] 0.9438521
```

```
## Improvement of adding regularization with lambda=6 is -5.660268e-05
```

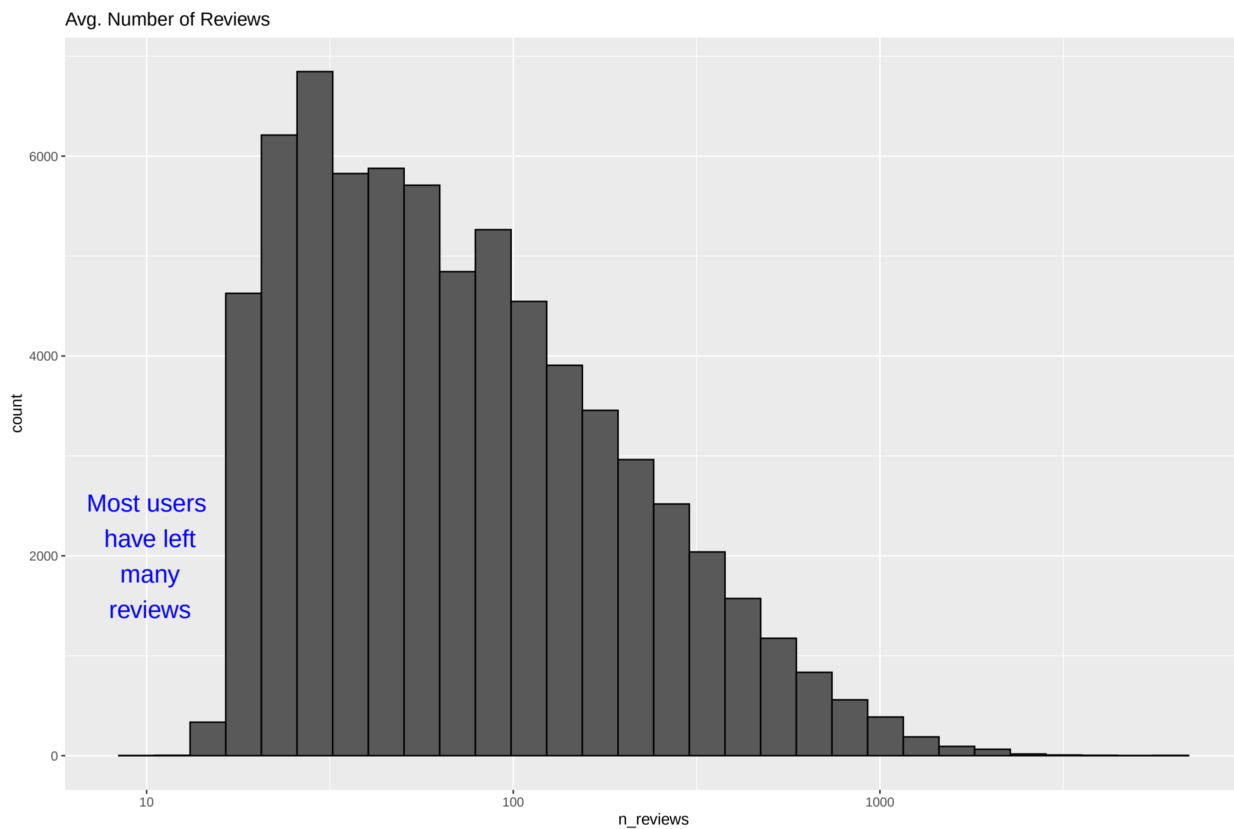
A small improvement again, but better than the arbitrary lambda!

Note that these incremental steps will be supressed for ease of reading. Refer to the r script for the detailed script.

User Effect

Next, users have their own personal bias. We can use each individuals average ratings as a predictor.

Let's look at the distribution of users:




```
## Minimum reviews left by a user = 10
```

To calculate user effect (aka bias), we take the mean of the different in movie ranking (bm) and overall average rank (mu)

userId	bu
1	1.68
2	-0.24
3	0.26
4	0.65
5	0.09
6	0.35

Join that back into the test data to predict ratings with movie bias AND user bias

```
## RMSE with user effect and movie regularized is 0.8652234
```

That's a huge improvement from modeling just the movie effect:

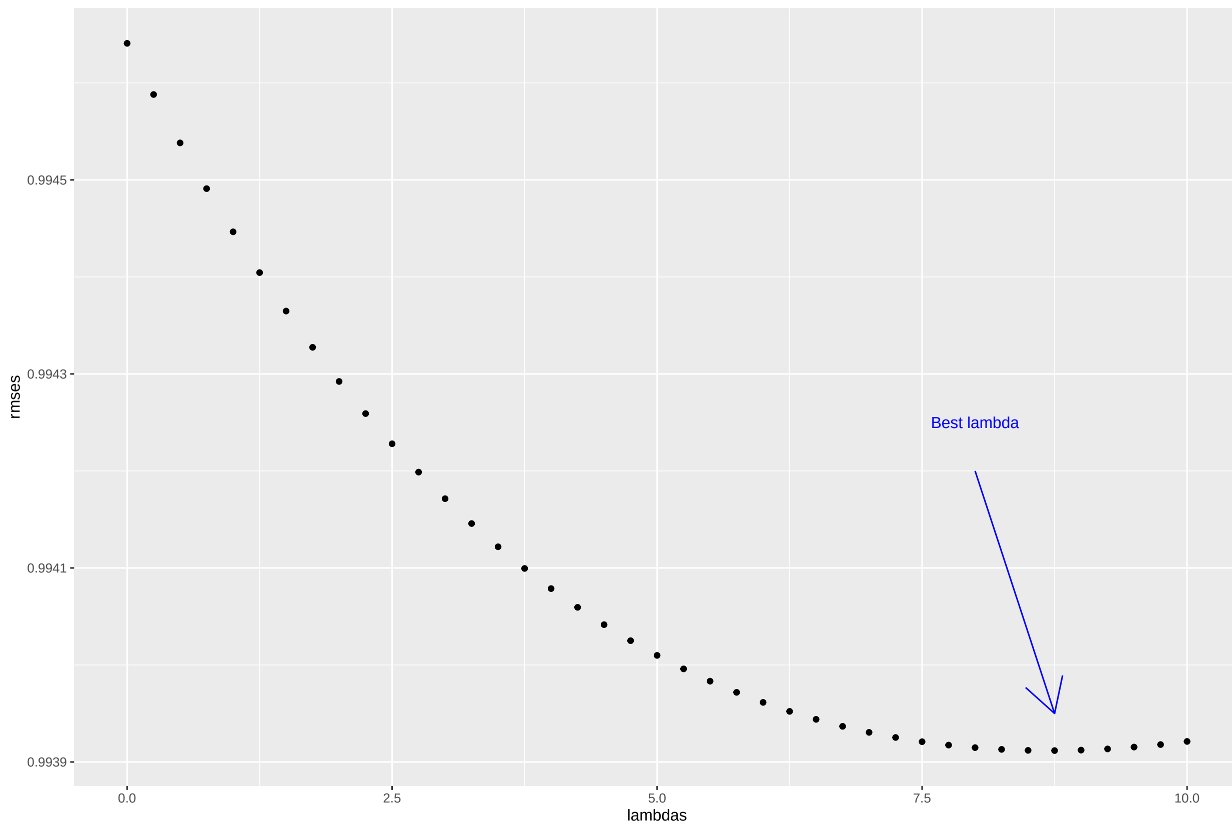
```
Naive_plus_reg_Movie_plus_User - Naive_plus_reg_Movie
```

```
## [1] -0.07862867
```

Should we do regularization? Looking at the most off predictions…

userId	resid_abs	n
9269	2.37	19
9586	2.28	28
16504	2.14	25
21871	2.21	30
22236	2.11	23
25958	2.42	83
47944	2.22	36
53601	2.14	20
58191	2.10	17
59073	2.16	16

… we see that the largest mis-predictions come from users with few reviews. We will use regularization again here to improve RMSE.




```
## Best lambda to minimize RMSE: 8.75
```

Now we use the best lambda to make our latest predictions:

```
## [1] 0.8649492
```

Naive_plus_reg_Movie_plus_User	Naive_plus_reg_Movie_plus_reg_User
0.8652234	0.8649492

```
## Adding Regularization to the User effect improved accuracy by -0.0002742337
```

Again, another improvement on our estimate.

Genre Effect

We next add the genre of the movies into the model. As shown earlier, there are many unique genres as movies can hve a combination of many.

```
## Total unique genres = 797
```

genres	bg	n
Drama	0.0225235	733296
Comedy	-0.0023361	700889
Comedy|Romance	-0.0108807	365468
Comedy|Drama	0.0238100	323637
Comedy|Drama|Romance	0.0061190	261425
Drama|Romance	0.0104538	259355
Action|Adventure|Sci-Fi	-0.0177115	219938
Action|Adventure|Thriller	-0.0277026	149091
Drama|Thriller	0.0032169	145373
Crime|Drama	0.0243277	137387

Inputting the adjustor for genre (bg) into our prediction model, we get a slightly better RMSE:

```
## [1] 0.8646053
```

```
## Improvement to RMSE is -0.0003438649
```

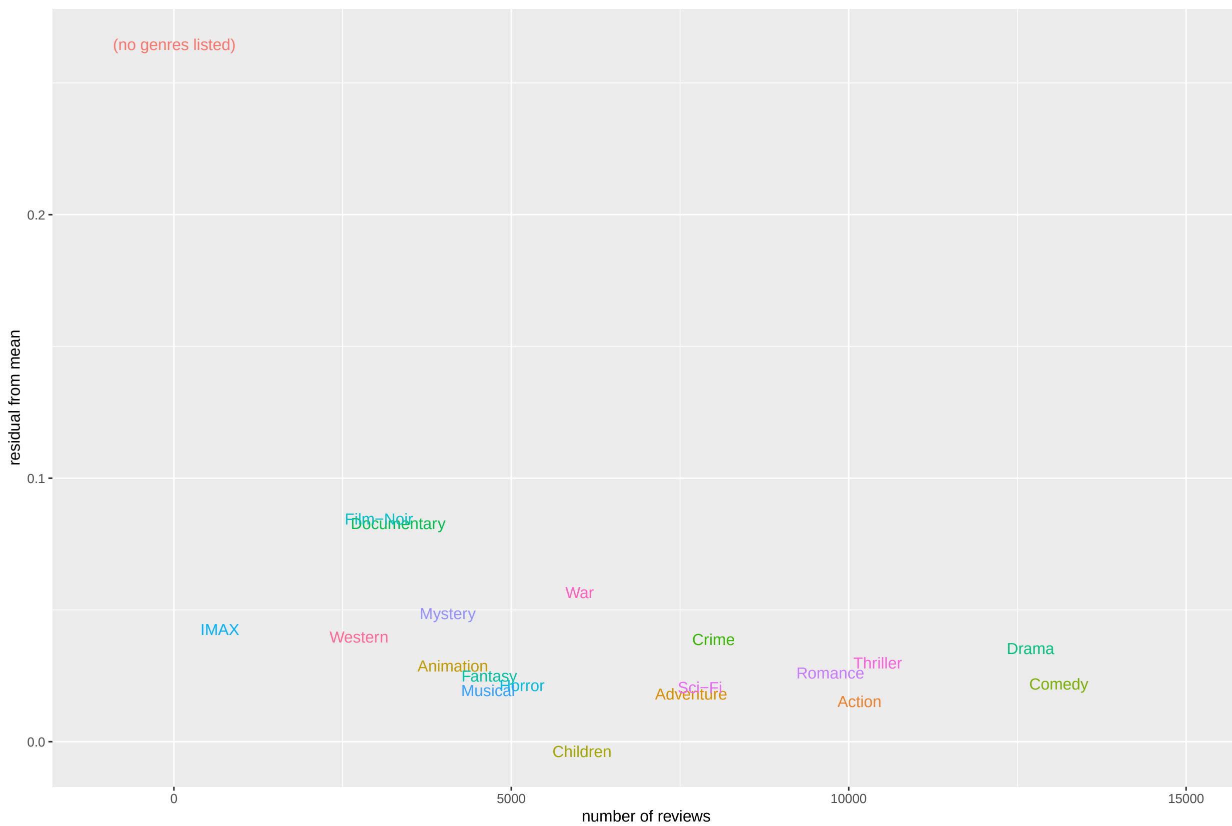
That's not a great improvement. The places where we are most off are on the genres with few reviews.

genres	avg_r	avg_p	delta	n
Crime|Film-Noir|Romance	0.50	3.56	-3.06	1
Animation|Horror|IMAX	1.50	3.01	-1.51	1
Action|Fantasy|Romance	2.33	3.75	-1.42	3
War|Western	1.75	3.13	-1.38	2
Action|Adventure|Animation|Comedy|Sci-Fi	2.50	3.77	-1.27	1
Children|Fantasy|Musical|Romance	0.50	1.76	-1.26	1
Animation|Fantasy|Sci-Fi|War	2.33	3.54	-1.21	3
Action|Drama|Fantasy	1.50	2.64	-1.14	2
Thriller|Western	2.67	3.69	-1.03	3
Action|Crime|Drama|Film-Noir|Thriller	2.50	3.47	-0.97	2

However, instead of using regularization, we can use what we know about each individual genre. We break up the genres field into unique, single genres with the below code:

```
indi_genre <- edx %>% left_join(movie_reg_avgs) %>%
  left_join(user_reg_avgs)  %>% group_by(genres) %>%
  summarise(n = n(), bg = mean(rating - (mu + bu + bi))) %>%
  separate_rows(genres, sep = "\\|") %>% # separate into rows
  group_by(genres) %>% summarise(bg = mean(bg), n=mean(n))
```

genres	bg	n
(no genres listed)	0.26	7.00
Action	0.02	10160.89
Adventure	0.02	7666.23
Animation	0.03	4134.23
Children	0.00	6049.13
Comedy	0.02	13114.56
Crime	0.04	7998.28
Documentary	0.08	3323.79
Drama	0.04	12695.22
Fantasy	0.02	4674.93
Film-Noir	0.08	3039.51
Horror	0.02	5160.34
IMAX	0.04	681.75
Musical	0.02	4656.77
Mystery	0.05	4059.51
Romance	0.03	9727.84
Sci-Fi	0.02	7797.58
Thriller	0.03	10430.04
War	0.06	6013.49
Western	0.04	2744.84



Every genre is liked except children…

Next, we pipe them back together, taking the mean of each indiviual genre when combined

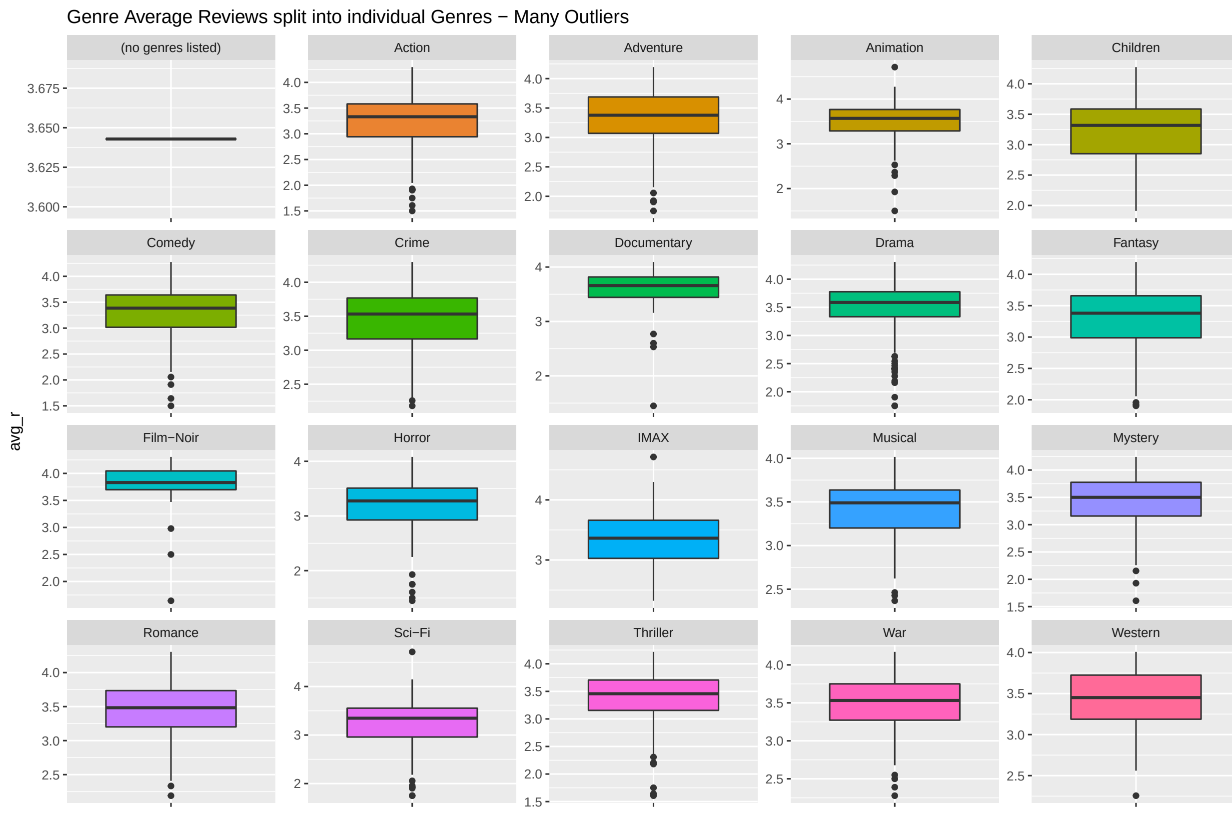
```
genre_lookup <- edx %>% select(genres) %>% group_by(genres) %>% summarise(n=n()) %>% #summarized to red
  mutate(g2 = genres) %>% separate_rows(genres, sep = "\\|") %>% left_join(indi_genre, by="genres") %>%
  group_by(g2) %>% summarise(bg = mean(bg)) %>% mutate(genres = g2) %>% select(genres, bg)

head(genre_lookup, 10) %>% kable(digits=3) %>% kable_styling(full_width=F)
```

genres	bg
(no genres listed)	0.265
Action	0.015
Action|Adventure	0.017
Action|Adventure|Animation|Children|Comedy	0.016
Action|Adventure|Animation|Children|Comedy|Fantasy	0.018
Action|Adventure|Animation|Children|Comedy|IMAX	0.020
Action|Adventure|Animation|Children|Comedy|Sci-Fi	0.017
Action|Adventure|Animation|Children|Fantasy	0.017
Action|Adventure|Animation|Children|Sci-Fi	0.016
Action|Adventure|Animation|Comedy|Drama	0.024

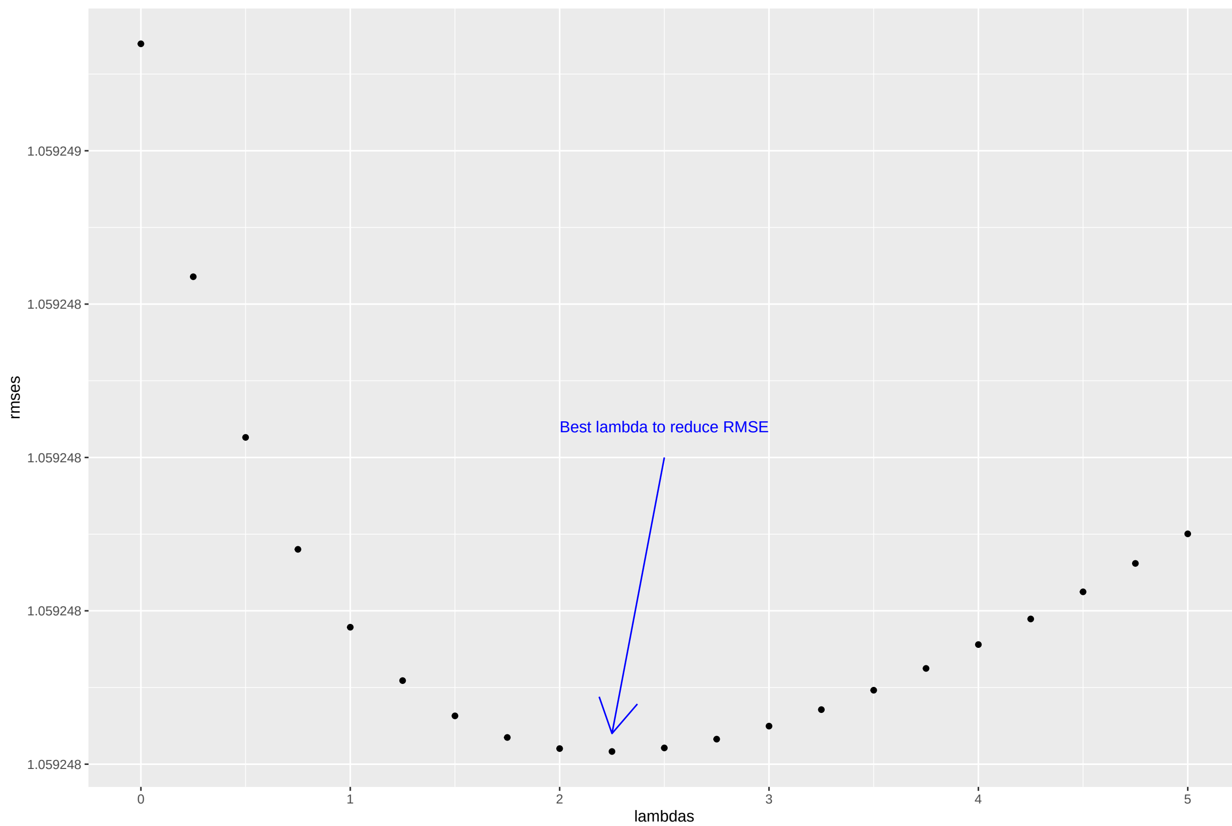
Now let's use this blended genre effect to estimate predictions:

```
## [1] 0.8651743
```

That did not help our model! Why? When looking at boxplots for average ratings at the individual genre level, we see many outliers. This is because the combination of genres in movies cause more randomness in reviews.



So we're scrapping the individual weighting approach and instead will use regularization.




```
## Best lambda to minimize RMSE: 2.25
```

```
## RMSE with Genre regularized is 0.8649492
```

```
## Improvement of using regularization instead of a individual weighted approach is -0.0002250988
```

Regularization works better for genres as well.

Year Effects

Lastly, we will use the year in which the movie was produced as a predictor. Some data wrangling will be necessary, as the year is stored within the movie title. # I will want to inspect year of movie as a predictor, so will separate

titles
Boomerang (1992)
Net, The (1995)
Outbreak (1995)
Stargate (1994)
Star Trek: Generations (1994)

We use regex and the stringr package to extract year:

```r
library(stringr)
year_pattern <- "\\([12][0-9][0-9][0-9]\\)"
str_detect(edx$title[1:5], year_pattern)
```

```
## [1] TRUE TRUE TRUE TRUE TRUE
```

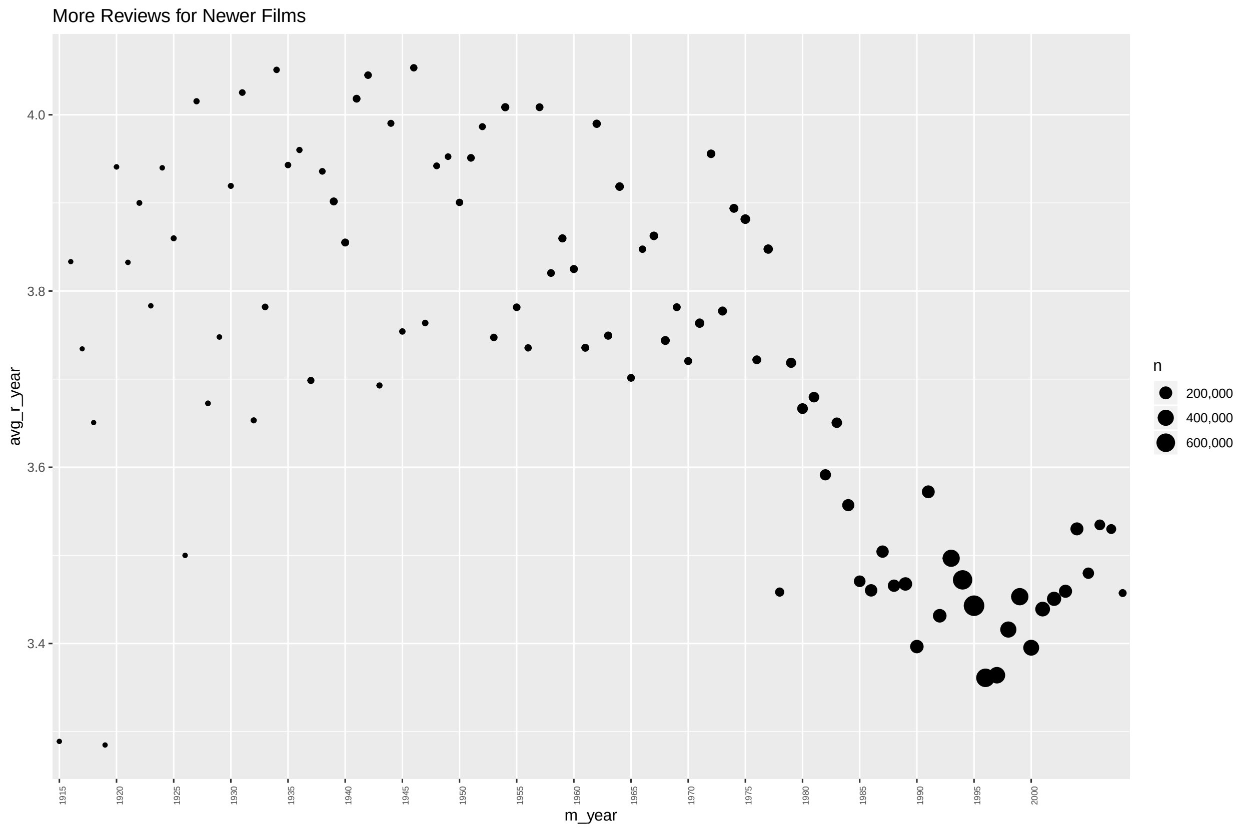
```r
str_view(as.character(edx$title[1:5]), year_pattern)
```

Boomerang (1992)
Net, The (1995)
Outbreak (1995)
Stargate (1994)
Star Trek: Generations (1994)

```r
## apply this to train and test sets and split out title name
#  apply this to the test and train groups
edx <- edx  %>% mutate(m_year = substr(str_extract(title, year_pattern), 2, 5), #skip the parentheses
                       title_clean = substr(title, 1, str_locate(title, year_pattern)[,1] - 2)) # remov
validation <- validation %>% mutate(m_year = substr(str_extract(title, year_pattern), 2, 5),
                                    title_clean = substr(title, 1, str_locate(title, year_pattern)[,1] 

kable(edx %>% select(title, title_clean, m_year) %>% slice(1:5)) %>% kable_styling(full_width = F)
```

title	title_clean	m_year
Boomerang (1992)	Boomerang	1992
Net, The (1995)	Net, The	1995
Outbreak (1995)	Outbreak	1995
Stargate (1994)	Stargate	1994
Star Trek: Generations (1994)	Star Trek: Generations	1994

Now lets inspect the yearly data:




```
## Fewest reviews are in year 1917 with only 32 reviews.
```

We see lots of variability in the earlier years with less reviews made. However, regularization is not a good option here as there are ample reviews across the years.

```
## Best lambda to minimize RMSE: 0
```

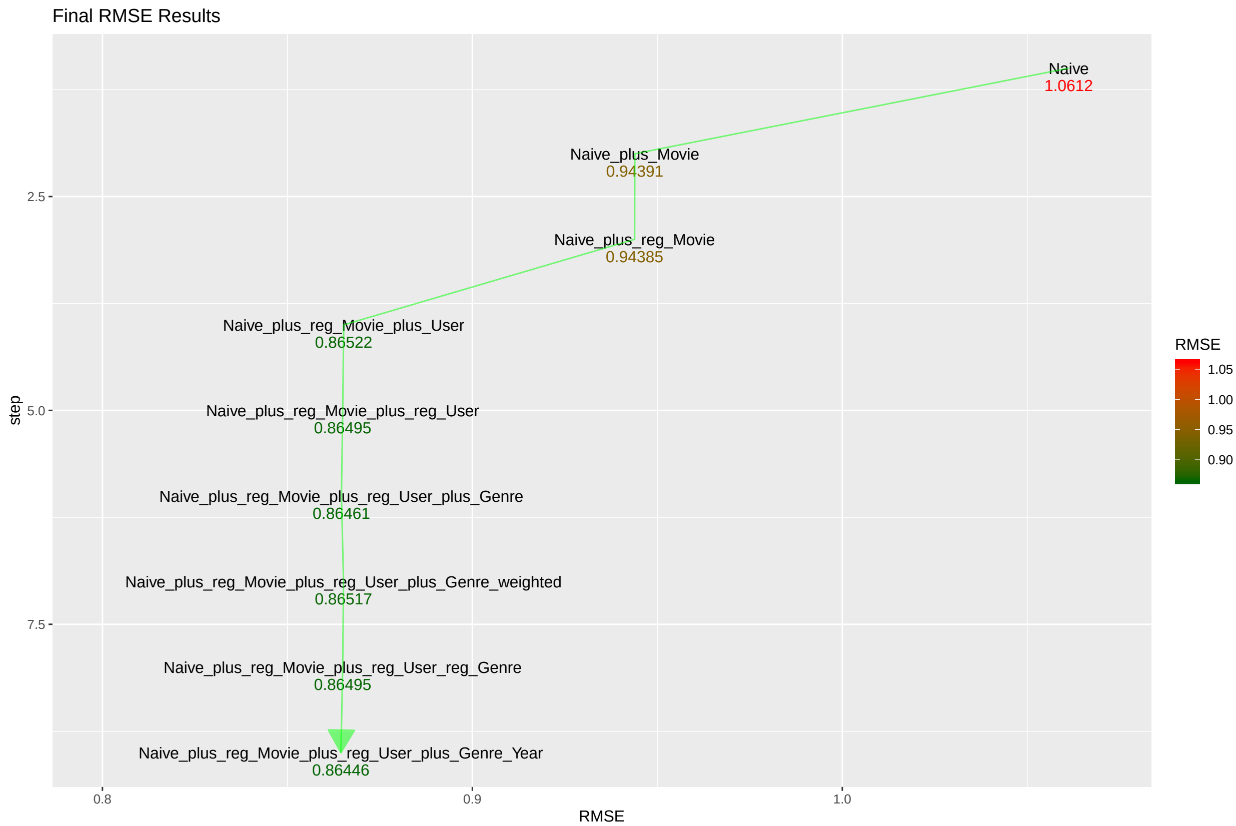
Regularization is not ideal for the year effect, so we will apply the yearly average as a predictor (by).

```
## Prediction with year effect included yields a final RMSE of 0.8644603
```

3. Results

Our final model uses regularized Movie, User, and Genre effects as well as Yearly effects. Below, you can see the comparison of the 9 different models built:

	RMSE Results
Naive	1.0612018
Naive_plus_Movie	0.9439087
Naive_plus_reg_Movie	0.9438521
Naive_plus_reg_Movie_plus_User	0.8652234
Naive_plus_reg_Movie_plus_reg_User	0.8649492
Naive_plus_reg_Movie_plus_reg_User_plus_Genre	0.8646053
Naive_plus_reg_Movie_plus_reg_User_plus_Genre_weighted	0.8651743
Naive_plus_reg_Movie_plus_reg_User_reg_Genre	0.8649492
Naive_plus_reg_Movie_plus_reg_User_plus_Genre_Year	0.8644603



4. Conclusion

After incrementing new effects into the model, we found that adding regularization helped improve the Root Mean Square Error. Unfortunately, using the weighted genre calculation was not effective as its impact was skewed due to randomness in the ratings. We also found that regularization did not improve results for effects with large numbers of records such as year.

A limitation with this Recommendation Model is that new recommendations assume that there is previous data on the user and the movie. The control group encompassed all movies and users in the test group.

Thanks for reading!