MovieLens Project - Another Look at The Netflix Prize

David List

03/06/2023

1 Introduction

"The Netflix Prize" was a contest announced by Netflix in October 2006 geared towards improving the accuracy of their movie recommendation system, Cinematch. The contest was simple in concept, though difficult in practice. Improve upon the accuracy of Cinematch by 10% and win a million dollars. In support of the contest, Netflix also released a dataset of over 100 million movie ratings from customers of their business which at the time was largely DVD movie rentals. They did not introduce their streaming service until later in February 2007. The contest lasted until September 2009 when it was won by team "BellKor Pragmatic Chaos" after they finally crossed the 10% threshold. Interestingly enough, Netflix never implemented the full winning solution in their system for a variety of reasons, including the shift in their business from DVDs to streaming. Note, however, they did include other improvements discovered over the nearly 3 years of the contest.

"The Netflix Prize" serves as one of the inspirations of this project, the other, of course, being the series of edX Data Science courses put together by Prof. Rafael Irizarry of Harvard University of which this project is one of two final assignments. While The Netflix Prize serves as an inspiration, the dataset analyzed for this project, however, is not the original Netflix data. It is instead the MovieLens 10M Dataset from GroupLens Research, a research lab associated with the University of Minnesota. Other than consisting of similar data, it has no apparent relationship to the original Netflix dataset, as that is not publicly available. The MovieLens dataset consists of data collected from the movielens.org website, itself a movie recommendation system, operated by GroupLens.

The purpose of this project, similar to the purpose of "The Netflix Prize," is to develop a machine learning algorithm to predict, as accurately as possible, the movie ratings contained in a "final holdout test" set utilizing data from a separate "training" set. In this case the final holdout test and training sets are generated from the sample code provided for that purpose by edX.

In developing an algorithm, the algorithm demonstrated by Prof. Irizarry in Section 6.2, of the edX course, "Data Science: Machine Learning" is used as the starting point. Much of the same material is also covered in Sections 33.7-33.10 of the book accompanying the course. In both sets of material, Prof. Irizarry develops an algorithm utilizing the following components:

- Average Movie Rating (mu)
- Average Rating by Movie with Regularization (bi_r) or Movie Effects
- Average Rating by User with Regularization (bu_r) or User Effects

Prof. Irizarry refers to this strategy as Matrix Factorization albeit a version that doesn't include PCA (Principal Component Analysys) or SVD (Singular Value Decomposition.) Edwin Chen of Surge AI describes this strategy as Normalization of Global Effects in his overview of the various techniques used to win the Netflix Prize. At any rate, the algorithm presented here starts with Prof. Iriazarry's algorithm and then expands it with the following additional effects:

- Average Rating by Movie Genre (bg) or Genre Effects

- Average Rating by User and Genre with Regularization (bgu_r) or User/Genre Effects
- Timeline Effects (bt)
- Ratings per Movie Effects

Note that each of these additional effects are suggested, though not developed, in the Exercises from Section 33.8 the book from Prof. Irizarry's course. In this project, each is developed into a component of a working model and discussed in detail in the following sections.

2 Analysis

2.1 Data Cleaning

No data cleaning was required for the project as the data is already fairly pristine.

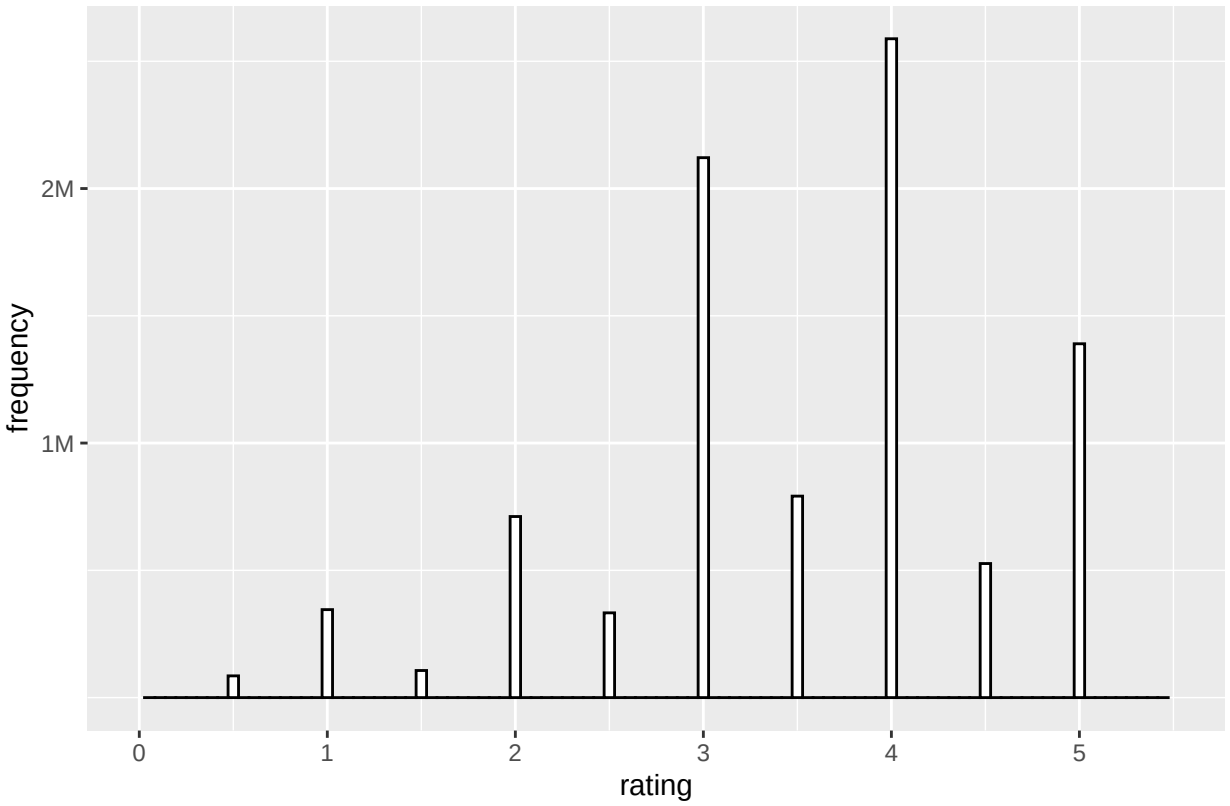
2.2 Data Exploration

The following sections provide different views of the MovieLense dataset to help provide a better understanding of the data. Note the dataset used in the course is a much smaller version of the one used for this project.

2.2.1 Ratings Histogram

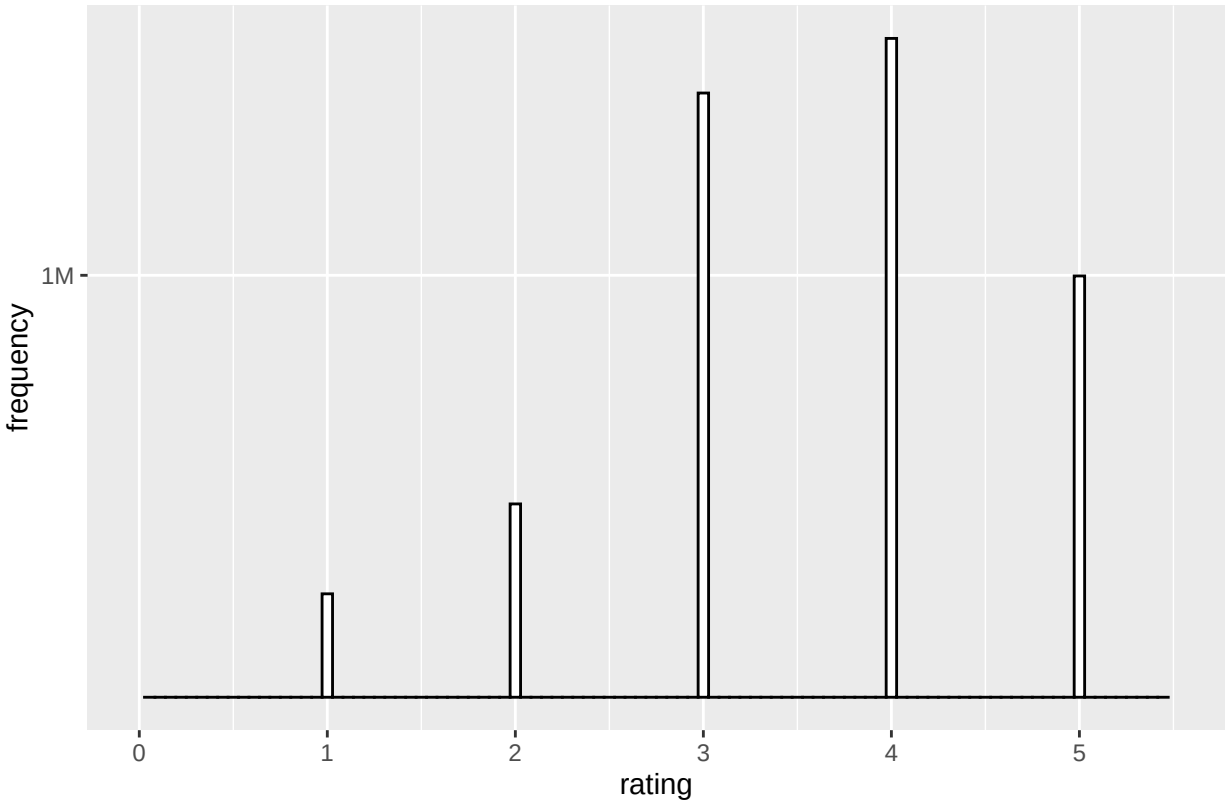
This first figure illustrates the set of values available to users to rate different movies as well as the frequency with which each value was chosen.

Figure 1: Ratings Histogram



One potential interpretation of Figure 1 is that the users appear to have a significant bias towards rating movies with whole numbers even though half numbers are clearly available. However, a closer look at the data reveals this is not the case at all. The first half number rating is timestamped February 12, 2003. As we will see in the next section, this is roughly half way through the period covered by the dataset. The following figures display separate histograms for the periods before and after February 12, 2003.

Figure 2: Ratings Histogram Before February 12, 2003



As indicated by the title, Figure 2 displays a histogram of the movie ratings before February 12, 2003. One interesting observation is that while a rating of 4 is still the most poplular movie rating, the frequency of ratings of 3 is much closer than that in Figure 1.

Figure 3: Ratings Histogram February 12, 2003 and After

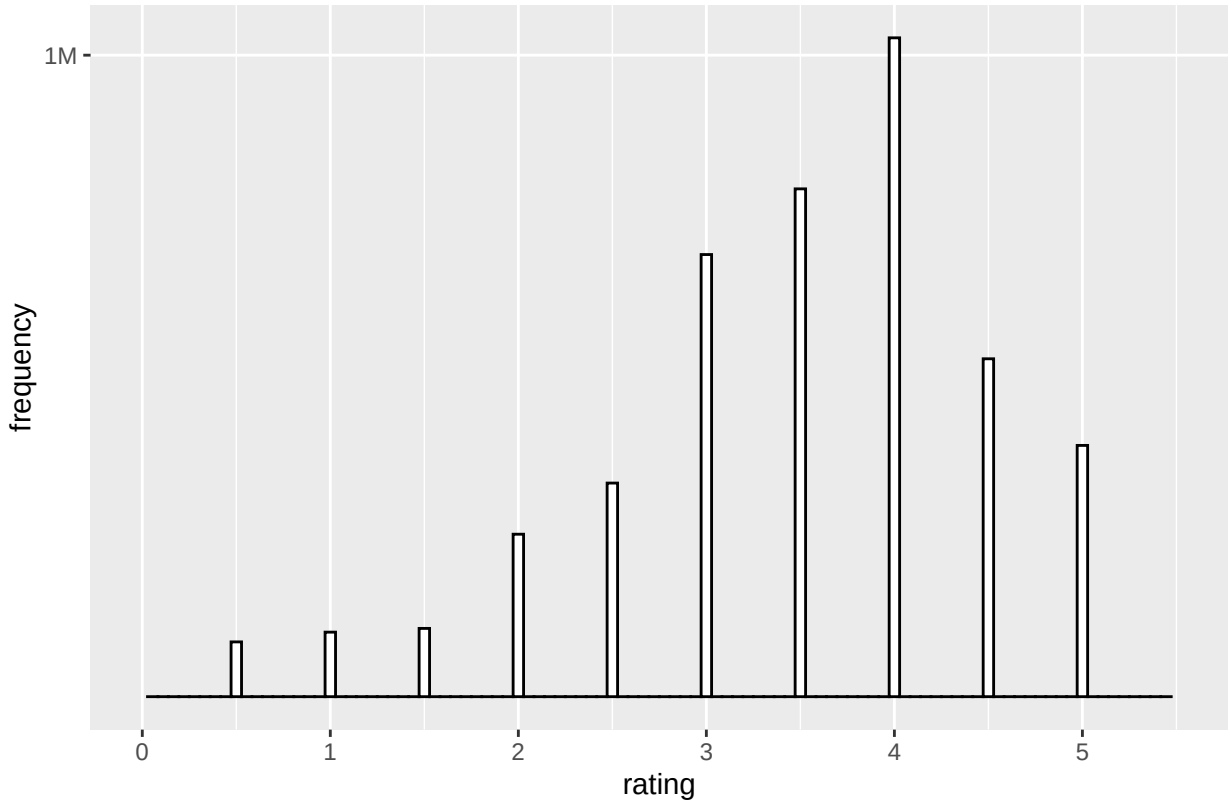


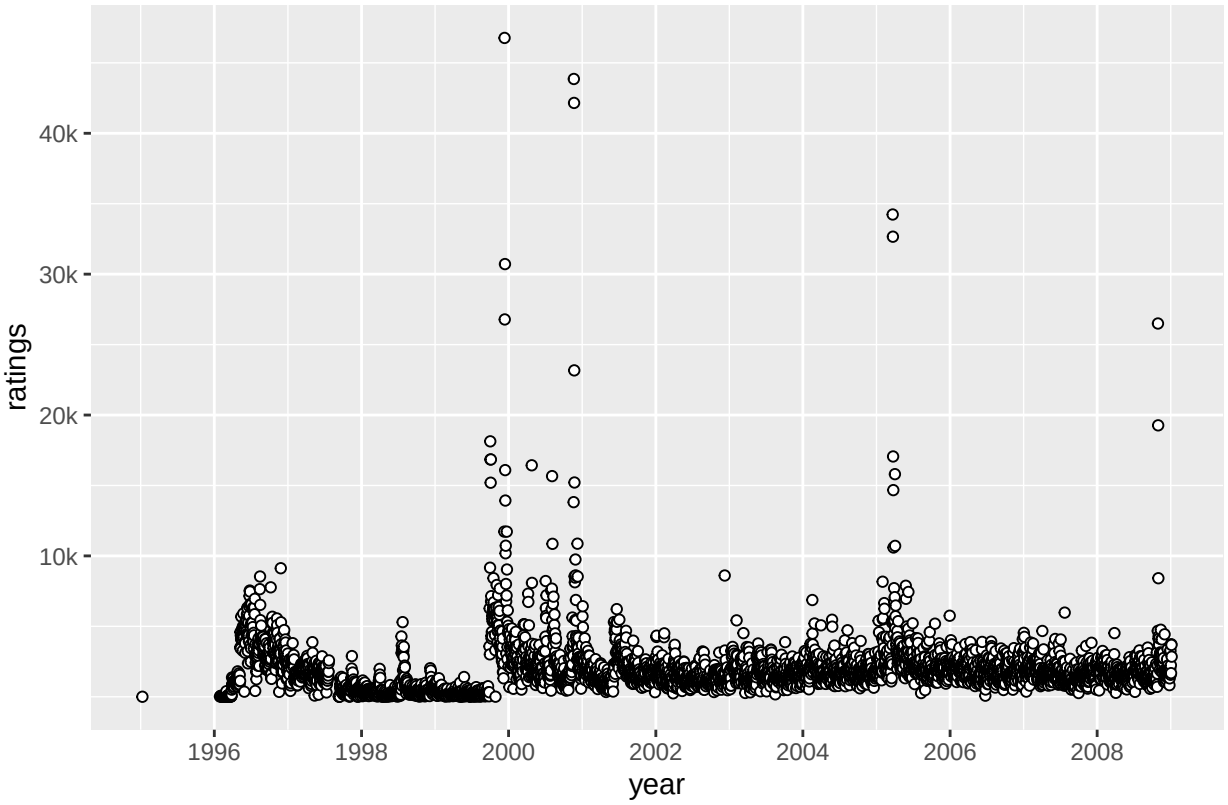
Figure 3 displays a histogram of the the movie ratings including February 12 and after. A section of a blog post on GroupLens.org suggests the reason for the change in available ratings:

"One result of pulling together 17 years of statistics is a surprising consistency in member rating behavior. For instance, we might expect that the major redesigns of the MovieLens ratings widget — including a shift from star ratings to half-star ratings in 2003 — to skew the contributions of ratings over time. Indeed, 17 years of user turnover, 17 years of changing user expectations concerning movies and online technologies, and 17 years of web site changes has led to a surprisingly consistent distribution of rating values stored in the MovieLens database."

2.2.2 Ratings over Time

Figure 4 displays the daily frequency of movie ratings over the full period of the dataset.

Figure 4: Ratings per Day



A few interesting observations here. First, there are a couple of ratings in early 1995 – it's actually 2 on January 9th as we'll see in the next section – and then nothing for about a year. Second, every now and then something occurs that causes activity to spike to many times above normal, such as in late 1999 and 2000. Finally, there appears to be no significant change in activity related to the ratings change from February 2003 mentioned in the previous section. The following table includes additional useful statistics:

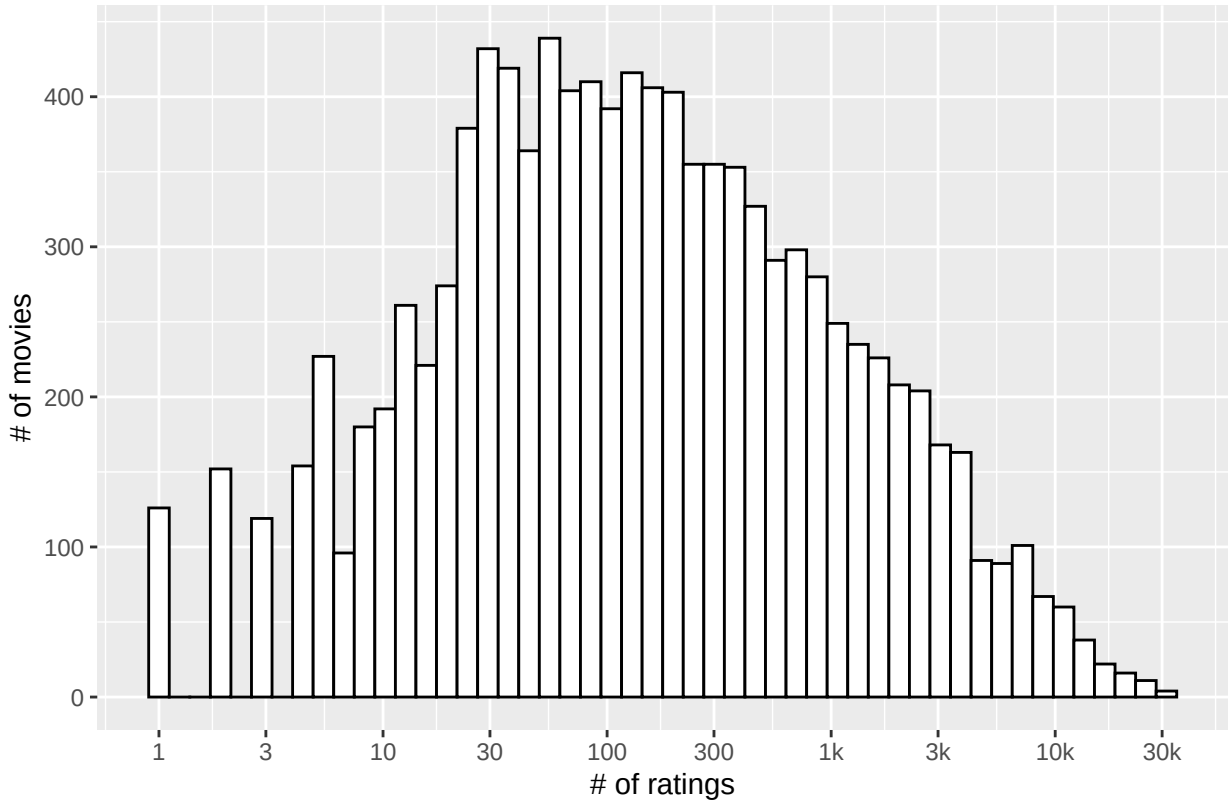
Attribute	Value
Start Date/Time	1995-01-09 11:46:49Z
End Date/Time	2009-01-05 05:02:16Z
Total Days	5111
Total Ratings	9000055
Average Ratings per Day	1761
Max Ratings per Day	46770 (12/12/1999)

Note these statistics are just for the edx portion of the dataset and do not include the final holdout test.

2.2.3 Movies by Number of Ratings

The following chart shows the distribution of movies by number of ratings.

Figure 5: Number of Movies vs. Number of Ratings

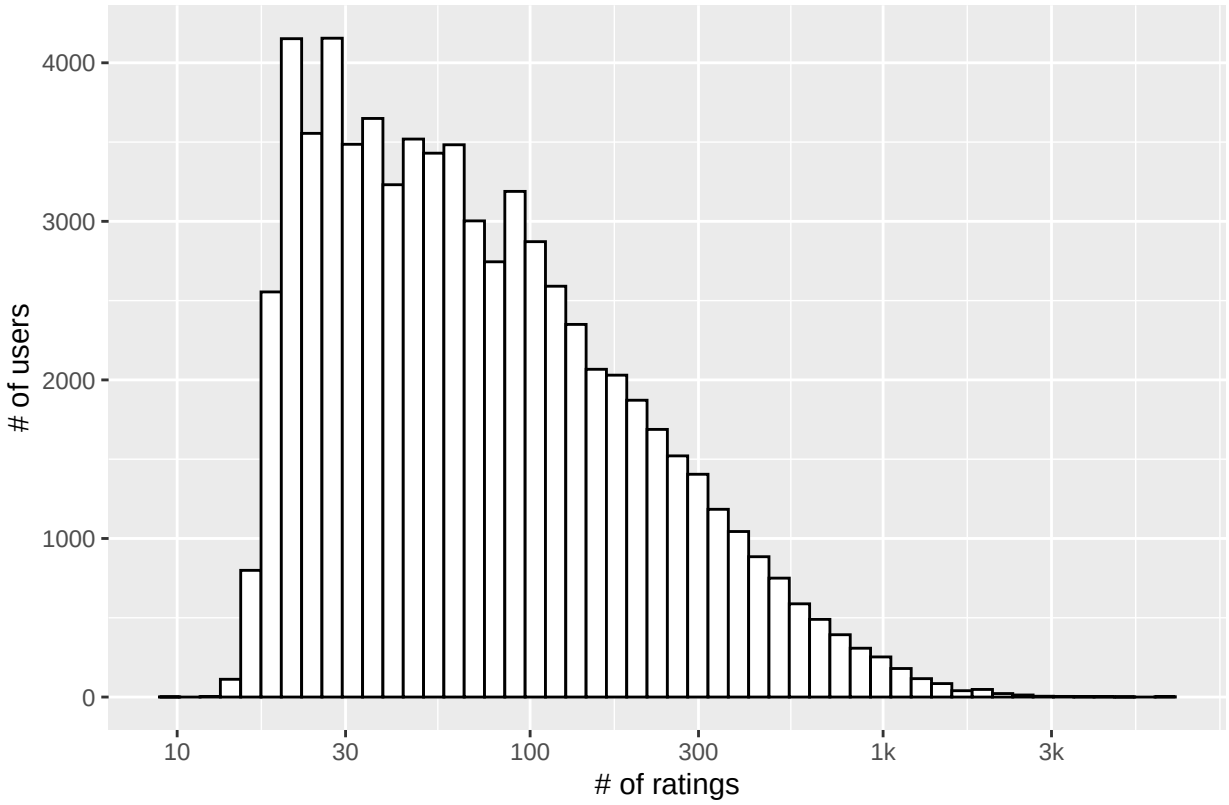


Attribute	Value
Total Movies	10677
Average Ratings per Movie	842

2.2.4 Users by Number of Ratings

The following chart shows the distribution of users by number of ratings.

Figure 6: Number of Users vs. Number of Ratings



Attribute	Value
Total Users	69878
Average Ratings per User	129

2.2.5 Sample Data

The following tables display the first and last 10 rows of data to introduce the format and fields of the data that make up a rating. Notice the different format of the rating between the two tables.

Table 1: The First 10 Rows

userId	movieId	rating	timestamp	title	genres
36955	47	5	789652009	Seven (a.k.a. Se7en) (1995)	Crime|Horror|Mystery|Thriller
36955	1079	3	789652009	Fish Called Wanda, A (1988)	Comedy|Crime
35139	1	4	822873600	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
35139	21	5	822873600	Get Shorty (1995)	Action|Comedy|Drama
35139	31	5	822873600	Dangerous Minds (1995)	Drama
35139	32	5	822873600	12 Monkeys (Twelve Monkeys) (1995)	Sci−Fi|Thriller
35139	39	5	822873600	Clueless (1995)	Comedy|Romance
35139	45	4	822873600	To Die For (1995)	Comedy|Drama|Thriller
35139	47	5	822873600	Seven (a.k.a. Se7en) (1995)	Crime|Horror|Mystery|Thriller
35139	52	4	822873600	Mighty Aphrodite (1995)	Comedy|Drama|Romance

Table 2: The Last 10 Rows

userId	movieId	rating	timestamp	title	genres
62510	34148	3.0	1231131736	Beat That My Heart Skipped, The (De battre mon coeur s'est arrêté) (2005)	Drama
62510	4784	2.5	1231131303	French Lieutenant's Woman, The (1981)	Drama
63100	2953	2.0	1231131142	Home Alone 2: Lost in New York (1992)	Children|Comedy
63100	7000	3.5	1231131137	Hudson Hawk (1991)	Action|Adventure|Comedy
63100	2478	3.5	1231131132	Three Amigos (1986)	Comedy|Western
63100	231	3.5	1231131127	Dumb & Dumber (1994)	Comedy
63100	410	3.5	1231131118	Addams Family Values (1993)	Comedy
63100	4816	3.5	1231131112	Zoolander (2001)	Comedy
63100	2361	4.5	1231131107	Pink Flamingos (1972)	Comedy
63100	36477	3.0	1231131103	Baxter, The (2005)	Comedy|Drama|Romance

2.2.6 Genres

The following list includes the set of genres that may be included as a pipe (|) delimited list in the genres field. Note that the first item, "no genres included," indicates an empty list.

- (no genres listed)
- Action
- Adventure
- Animation
- Children
- Comedy
- Crime
- Documentary
- Drama
- Fantasy

- Film-Noir
- Horror
- IMAX
- Musical
- Mystery
- Romance
- Sci-Fi
- Thriller
- War
- Western

2.2.6 Most Popular Genre Combinations

The dataset includes 797 combinations of the genres listed in the previous section across 10677 movies. The following table includes the 10 most popular combinations.

Genres	Count
Drama	733296
Comedy	700889
Comedy|Romance	365468
Comedy|Drama	323637
Comedy|Drama|Romance	261425
Drama|Romance	259355
Action|Adventure|Sci-Fi	219938
Action|Adventure|Thriller	149091
Drama|Thriller	145373
Crime|Drama	137387

2.2 Cross Validation

To save time K-fold Cross Validation is used with K equal to just one. Per the course material, typically values of 5 or 10 are used, but doing so increases the processing time by 5 and 10 times respectively which is prohibitive given the approximately 30 minutes just the one iteration takes. To validate using a value of 1, however, the models are run 2 additional times with different seed values at the end as a sanity check. While using a different seed value isn't perfect as it results in validation sets with slight overlap, the end result is a Cross Validation near K equal to 3.

To perform Cross Validation, the edx dataset is broken into two sets named edx_train and edx_test with the following code:

```
# Build new train/test sets so we don't use the final_holdout_test set
# until the very end.  Use 1.0/9.0 so edx_test is the roughly the
# same size as final_holdout_test.
set.seed(seed, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
edx_test_index <- createDataPartition(y = edx$rating, times = 1, p = 1.0/9.0,
                                      list = FALSE)
edx_train <- edx[-edx_test_index,]
edx_temp <- edx[edx_test_index,]

# As above, keep only the rows with users and movies that exist
# in the training set.
edx_test <- edx_temp %>%
  semi_join(edx_train, by = "movieId") %>%
```

```
  semi_join(edx_train, by = "userId")
edx_removed <- anti_join(edx_temp, edx_test)
edx_train <- rbind(edx_train, edx_removed)

rm(edx_test_index, edx_temp, edx_removed)
```

2.3 Random Guess

Before diving into the details of the model it's instructive as a thought exercise to consider the RMSE value that would result from guessing purely at random. This may be simulated with the following code:

```
# Calculate the approximate RMSE from randomly guessing.
random_guess <- sample(c(0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0),
                       nrow(edx_test), replace = TRUE)
y_hat <- random_guess
rmse_random <- rmse(y_hat, edx_test$rating)
```

Note that we calculate RMSE using the rmse() function which is defined as follows and used throughout this paper:

```
# Define a function for calculating for calculating rmse.
rmse <- function(y_hat, y) {
  sqrt(mean((y_hat - y)^2))
}
```

Variable	Calculated Value
Random Guess RMSE	1.94172

2.4 Average Rating (mu)

The next few sections closely follow the material from Prof. Irizarry in Section 6.2, of the edX course, "Data Science: Machine Learning". However, since Prof. Irizzary uses a much smaller dataset, it's interesting to compare results.

With that in mind, the first step is to calculate the average rating or (mu):

```
# Calculate the average rating from the training set, mu, and calculate the
# RMSE associated with always using mu.
mu <- mean(edx_train$rating)
y_hat <- mu
rmse_mu <- rmse(y_hat, edx_test$rating)
```

In this case the results agree pretty closely:

Variable	Course Value	Capstone Value
Average Rating (mu)	3.54	3.51247
Average Rating RMSE	1.05	1.06

2.5 Movie Effects (bi & bi_r)

Movie Effects (bi) are calculated next as the average movie rating for each movie after mu has been subtracted. This means values less than 3.54 are negative and values greater than 3.54 are positive.

The following code is used for this calculation:

```
# Calculate RMSE considering just movie effects.
bi <- edx_train %>%
  group_by(movieId) %>% summarize(bi = mean(rating - mu))
y_hat <- mu + edx_test %>% left_join(bi, by = 'movieId') %>% .$bi
rmse_bi <- rmse(y_hat, edx_test$rating)
```

Variable	Course Value	Calculated Value
Movie Effects RMSE	0.99	0.94311

Here the calculated results are about 5% better than the results from the course which seems significant given that the Netflix prize was awarded for only a 10% improvement. However, the difference is likely explained by the difference between the number of ratings vs. movies in the two datasets which is quite significant.

Dataset	Movies	Ratings	Ratio
Course Material (dslabs)	8469	80002	9.45
Capstone	10677	8000061	749.28

The next part of this section illustrates why differing ratios likely accounts for the differences in RMSE.

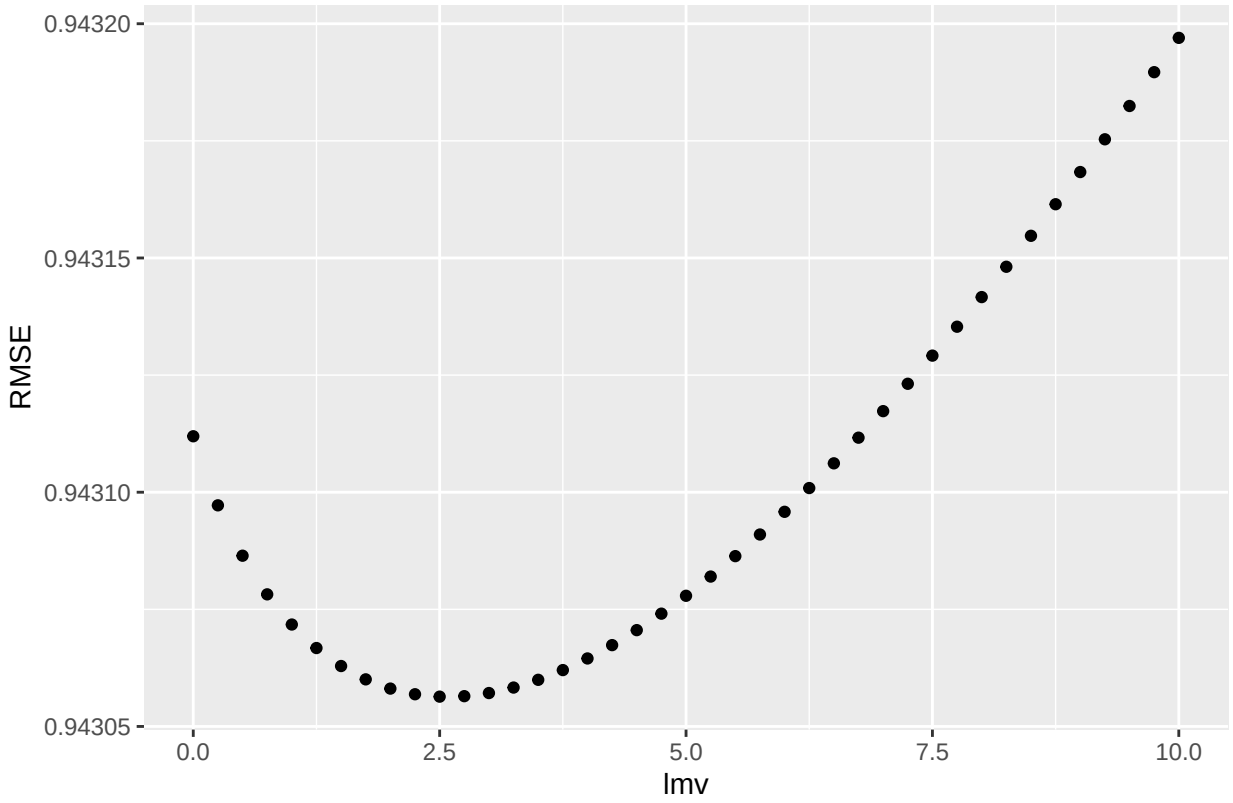
2.5.1 Regularization

Regularization is used to penalize the contribution of movies with a smaller number of ratings with the following code.

```
# Calculate rmse considering just movie effects plus regularization.
bi_r <- edx_train %>%
  group_by(movieId) %>%
  summarize(bi_r = sum(rating - mu)/(n() + lmv))
y_hat <- mu +
  edx_test %>% left_join(bi_r, by = 'movieId') %>% .$bi_r
rmse_bi_r <- rmse(y_hat, edx_test$rating)
```

Note the variable lmv in the above code. This is the regularization lambda, a tuning parameter, and is chosen by iterating through a series of values and selecting the one that results in the lowest RMSE. The following graph shows the results of that iteration process.

Figure 7: RMSE vs Lambda (lmv) for Movie Regularization



Variable	Course Value	Capstone Value
lmv	3	2.5
Movie Effects RMSE	0.99	0.94311
Movie Effects RMSE w/ Regularization	0.96	0.94306

The course and Capstone values for the selected lmv parameter agree pretty closely at 3 vs 2.5. However, with a ratio of 9.45 ratings per movie and a lambda of 3 for the course, regularization comes into play much more often than with a ratio of ~750 ratings per movie and a lambda of 2.5 for the Capstone dataset. This is further backed up by the fact that regularization accounts for about a 3% drop in RMSE for the course vs. almost no change for the Capstone dataset. (Note that the 0.96 value comes from rerunning the code from the course. The value displayed in the video was 0.88 which seems like a mistake.)

As a sanity check it makes sense to look at the effect regularization has on the ordering of the movies in the Capstone dataset. The following listing is the ordering from the unregularized bi value.

Title	bi
Hellhounds on My Trail (1999)	1.49
MC5*: A True Testimonial (2002)	1.49
Satan's Tango (Sátántangó) (1994)	1.49
Shadows of Forgotten Ancestors (1964)	1.49
Fighting Elegy (Kenka erejii) (1966)	1.49
Sun Alley (Sonnenallee) (1999)	1.49
Blue Light, The (Das Blaue Licht) (1932)	1.49
More (1998)	1.40
Human Condition II, The (Ningen no joken II) (1959)	1.24

Title	bi
Human Condition III, The (Ningen no joken III) (1961)	1.24

As expected these movies are quite obscure. Below is a similar listing using the regularized bi_r value.

Title	bi_r
More (1998)	0.991
Shawshank Redemption, The (1994)	0.943
Godfather, The (1972)	0.901
Usual Suspects, The (1995)	0.854
Schindler's List (1993)	0.852
Rear Window (1954)	0.812
Casablanca (1942)	0.808
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	0.802
Seven Samurai (Shichinin no samurai) (1954)	0.793
Third Man, The (1949)	0.790

And as with the corresponding listing from the course, these movies make a lot more sense. So even though regularization has almost no effect on RMSE in the Capstone dataset, it still has a substantial effect on the ordering of movie rankings.

2.6 User Effects (bu & bu_r)

The next step from the course is to consider User Effects (bu) so that is done here. User Effects are calculated as the average residual amount remaining after mu and bi_r have already been removed. Note that this is an important point. User effects are not calculated from the average user rating as might be expected. Like all of the effects, they are calculated from the remaining error or residual from the previous step.

The following code is used for this calculation:

```
# Calculate rmse additionally considering user effects.
bu <- edx_train %>%
  left_join(bi_r, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(bu = mean(rating - mu - bi_r))
y_hat <- mu +
  edx_test %>% left_join(bi_r, by = 'movieId') %>% .$bi_r +
  edx_test %>% left_join(bu, by = 'userId') %>% .$bu
rmse_bu <- rmse(y_hat, edx_test$rating)
```

Note that the course material doesn't calculate a value for Movie Effects w/ Regularization plus User Effects w/out Regularization as has been calculated here.

Variable	Course Value	Calculated Value
User Effects RMSE	—	0.86495

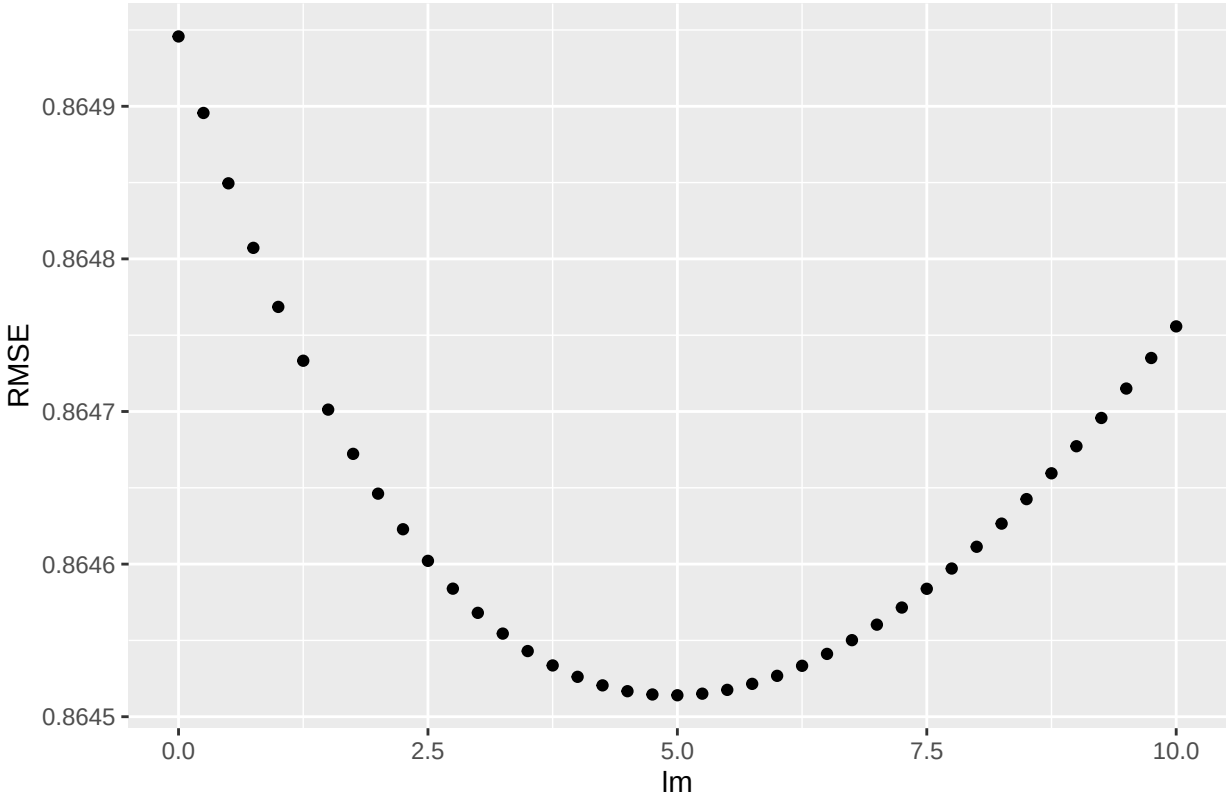
As suggested above regularization can also be performed for User Effects so that is done next. Unlike the course material, here a separate lambda value lu is calculated. In the course material a single lambda value is used for both User Effects and Movie Effects regularization.

The following code is used for this calculation:

```r
# Calculate rmse additionally considering user effects plus regularization.
bu_r <- edx_train %>%
  left_join(bi_r, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(bu_r = sum(rating - mu - bi_r)/(n() + lu))
y_hat <- mu +
  edx_test %>% left_join(bi_r, by = 'movieId') %>% .$bi_r +
  edx_test %>% left_join(bu_r, by = 'userId') %>% .$bu_r
rmse_bu_r <- rmse(y_hat, edx_test$rating)
```

As done previously, lu in the above code is chosen by iterating through a series of values and selecting the one that results in the lowest RMSE. The following graph shows the results of that iteration process.

Figure 7: RMSE vs Lambda (lu) for User Regularization



Variable	Course Value	Capstone Value
lu	3.75	5
User Effects RMSE	—	0.86495
User Effects RMSE w/ Regularization	0.881	0.86451

Note the difference between course and Capstone values has narrowed significantly from the original 5%.

At this point the course material moves on to new topics so the remaining sections build upon that original model. However, as mentioned in the introduction, the effects included in the following sections are all mentioned or implied as areas exploration in the Excercises from Section 33.8 of the book.

2.7 Genre Effects (bg)

This section explores Genre Effects which are calculated as the average residual amount remaining after mu, bi_r, and bu_r have been removed.

The following code is used for this calculation:

```
# Calculate rmse additionally considering genre effects.  Note the genres
# column includes zero to many genres separated by the pipe character.
# The following considers each combination of genres as a separate "genre"
# so for instance the effects of "Comedy", "Romance", and "Comedy|Romance"
# are considered separately.
bg <- edx_train %>%
  left_join(bi_r, by = 'movieId') %>%
  left_join(bu_r, by = 'userId') %>%
  group_by(genres) %>%
  summarize(bg = mean(rating - mu - bi_r - bu_r))
y_hat <- mu +
  edx_test %>% left_join(bi_r, by = 'movieId') %>% .$bi_r +
  edx_test %>% left_join(bu_r, by = 'userId') %>% .$bu_r +
  edx_test %>% left_join(bg, by = 'genres') %>% .$bg
rmse_bg <- rmse(y_hat, edx_test$rating)
```

Variable	Calculated Value
Genre Effects RMSE	0.86421

The effect is quite small but is interesting there is any effect at all. Conceptually Genre Effects seem like they should just be the average of the Movie Effects of the movies within a given genre, and since Movie Effects are already included, Genre Effects seem like they should be redundant. In theory regularization seems like it might cause small residuals to remain from the Movie Effects so perhaps this effect is related to that.

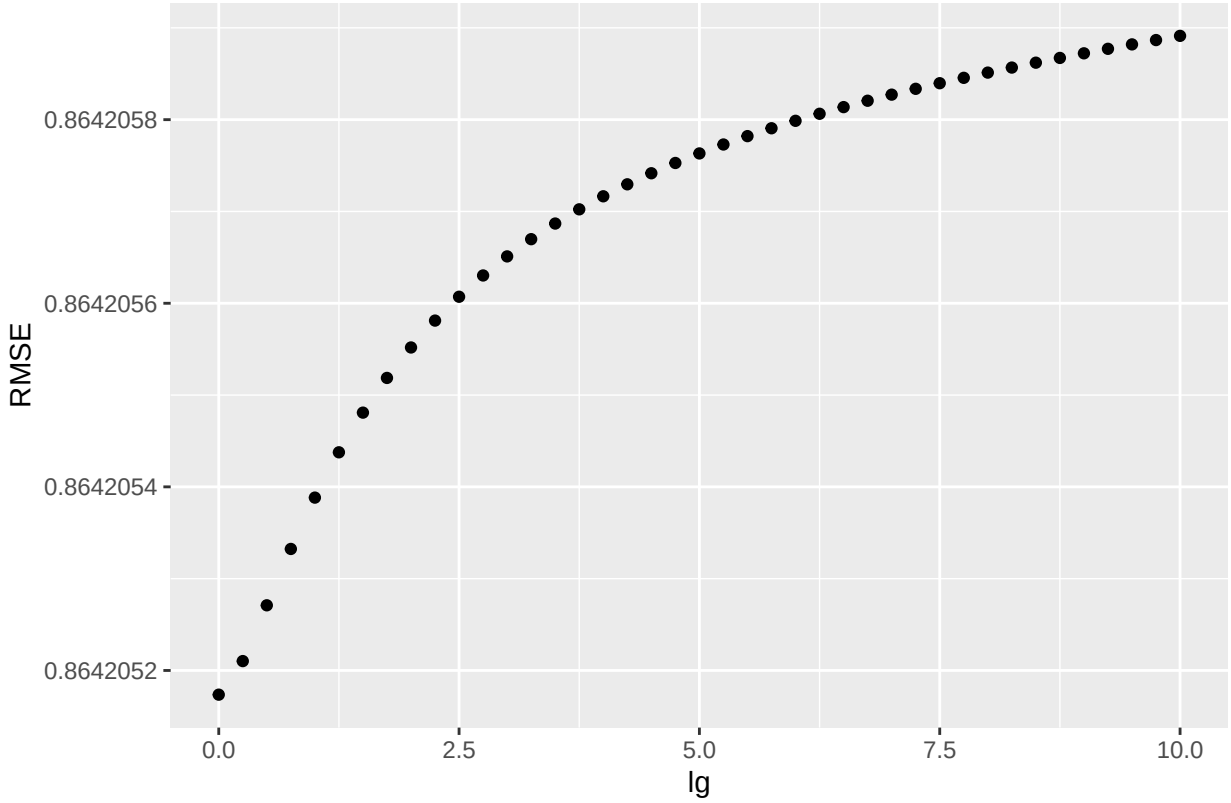
Similar to previous sections it makes sense to see if additional improvement can be found using regularization.

The following code is used for this calculation. Note this code is not included in the accompanying script. The reason for that will become clear below.

```
bg_r <- edx_train %>%
  left_join(bi_r, by = 'movieId') %>%
  left_join(bu_r, by = 'userId') %>%
  group_by(genres) %>%
  summarize(bg_r = sum(rating - mu - bi_r - bu_r)/(n() + lg))
y_hat <- mu +
  edx_test %>% left_join(bi_r, by = 'movieId') %>% .$bi_r +
  edx_test %>% left_join(bu_r, by = 'userId') %>% .$bu_r +
  edx_test %>% left_join(bg_r, by = 'genres') %>% .$bg_r
rmse_bg_r <- rmse(y_hat, edx_test$rating)
```

As done previously, lg in the above code is chosen by iterating through a series of values and selecting the one that results in the lowest RMSE. The following graph shows the results of that iteration process.

Figure 8: RMSE vs Lambda (lg) for Genre Regularization



Variable	Calculated Value
lg	0
Genre Effects RMSE	0.86421
Genre Effects RMSE w/ Regularization	0.86421

As seen from the graph no benefit is gained by using regularization with Genre Effects so it is not included in the model.

2.8 User/Genre Effects

This section explores User/Genre Effects which are calculated as the average residual amount for a given user/genre combination remaining after mu, bi_r, bu_r, and bg have already been removed.

The following code is used for this calculation:

```
# Calculate rmse additionally considering user specific genre effects.
bgu <- edx_train %>%
  left_join(bi_r, by = 'movieId') %>%
  left_join(bu_r, by = 'userId') %>%
  left_join(bg, by = 'genres') %>%
  group_by(genres, userId) %>%
  summarize(bgu = mean(rating - mu - bi_r - bu_r - bg))
y_hat <- mu +
  edx_test %>% left_join(bi_r, by = 'movieId') %>% .$bi_r +
  edx_test %>% left_join(bu_r, by = 'userId') %>% .$bu_r +
  edx_test %>% left_join(bg, by = 'genres') %>% .$bg +
```

```
  edx_test %>% left_join(bgu, by = c('genres', 'userId')) %>%
  # Replace N/A with 0 when user has no ratings for a given genre.
  mutate(bgu = ifelse(is.na(bgu), 0, bgu)) %>% .$bgu
rmse_bgu <- rmse(y_hat, edx_test$rating)
```

Variable	Calculated Value
User/Genre Effects RMSE	0.91847

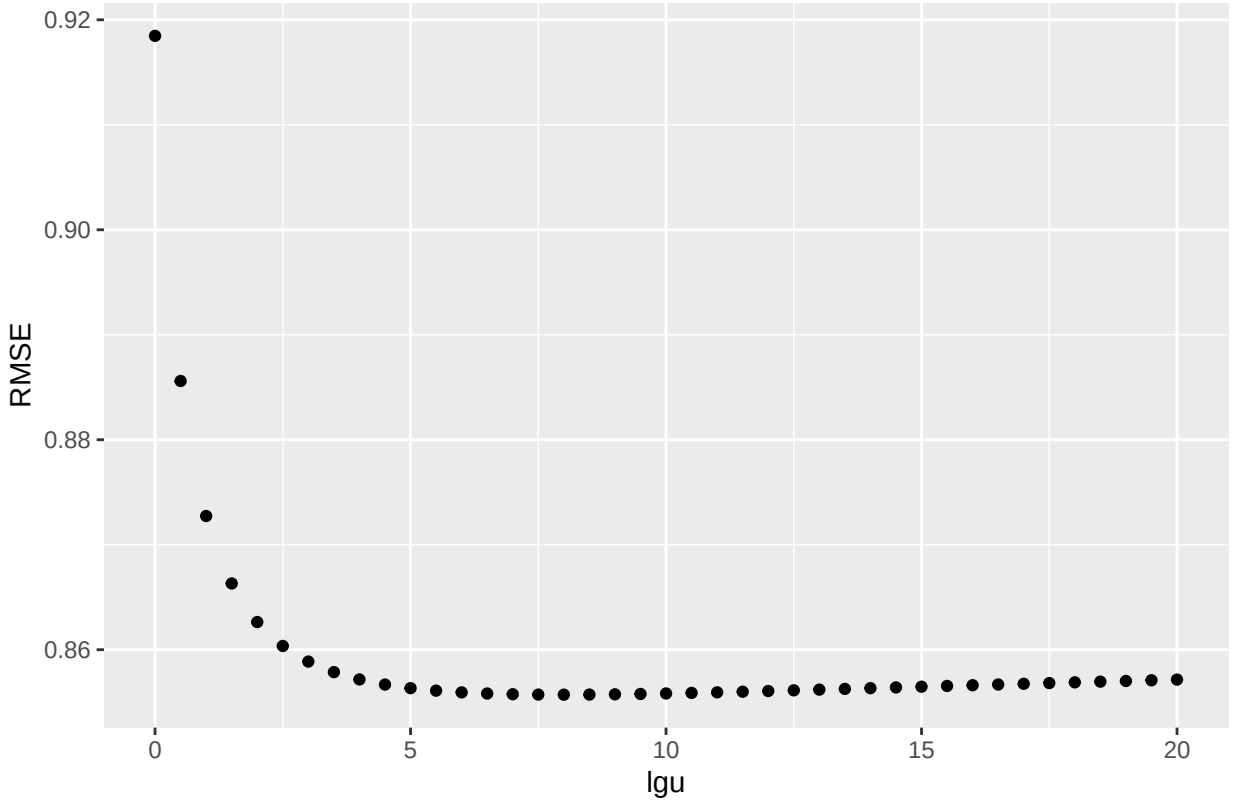
This result is disappointing as it substantially increases the RMSE value from previous sections. However, with a total of 797 genre combinations in the dataset and an average of 129 ratings per user this seems like a good candidate for regularization which is explored next.

The following code is used to calculate User/Genre Effects with Regularization:

```
# Calculate rmse additionally considering user specific genre effects plus
# regularization.
bgu_r <- edx_train %>%
  left_join(bi_r, by = 'movieId') %>%
  left_join(bu_r, by = 'userId') %>%
  left_join(bg, by = 'genres') %>%
  group_by(genres, userId) %>%
  summarize(bgu_r = sum(rating - mu - bi_r - bu_r - bg)/(n() + lgu))
y_hat <- mu +
  edx_test %>% left_join(bi_r, by = 'movieId') %>% .$bi_r +
  edx_test %>% left_join(bu_r, by = 'userId') %>% .$bu_r +
  edx_test %>% left_join(bg, by = 'genres') %>% .$bg +
  edx_test %>% left_join(bgu_r, by = c('genres', 'userId')) %>%
  mutate(bgu_r = ifelse(is.na(bgu_r), 0, bgu_r)) %>% .$bgu_r
rmse_bgu_r <- rmse(y_hat, edx_test$rating)
```

As done previously, lgu in the above code is chosen by iterating through a series of values and selecting the one that results in the lowest RMSE. The following graph shows the results of that iteration process.

Figure 9: RMSE vs Lambda (lgu) for User/Genre Regularization



Variable	Calculated Value
lgu	8
User/Genre Effects RMSE	0.91847
User/Genre Effects RMSE w/ Regularization	0.85572

These result are obviously much better. Also, it's important to note this change is the first one that actually changes the order of movies for different users. All the models from the previous sections provide the same top 10 list just with different values for the ratings.

To see this, the users with the largest effects for the Drama and Comedy genres are first selected.

The following list shows the top 10 users for "Drama":

genres	userId	bgu_r
Drama	15230	0.8268765
Drama	19679	0.8158130
Drama	45516	0.7958139
Drama	11760	0.7848641
Drama	60124	0.7585018
Drama	26353	0.7095548
Drama	71369	0.6865603
Drama	37380	0.6747518
Drama	53287	0.6672954
Drama	41601	0.6606894

This next list shows the top 10 users for "Comedy":

genres	userId	bgu_r
Comedy	25958	1.6628754
Comedy	41008	1.0400985
Comedy	28361	1.0393575
Comedy	20913	1.0183404
Comedy	10195	1.0109546
Comedy	65303	0.9236017
Comedy	14427	0.9066747
Comedy	42696	0.8933690
Comedy	48635	0.8767331
Comedy	8140	0.8667373

Next a "blank" movie list is created for each user and passed through the model the same way as the edx_test set would be.

The following code shows how this is done for the Drama fan for the model from the previous section (2.7):

```
drama_fan <- edx_train %>%
  mutate(userId=15230) %>%
  group_by(movieId) %>% summarize(userId=unique(userId),
                                  rating=0,
                                  timestamp=0,
                                  title=unique(title),
                                  genres=unique(genres))
drama_fan %>%
  mutate(rating = mu +
           drama_fan %>%
           left_join(bi_r, by = 'movieId') %>% .$bi_r +
           drama_fan %>% left_join(bu_r, by = 'userId') %>% .$bu_r +
           drama_fan %>% left_join(bg, by = 'genres') %>% .$bg) %>%
  arrange(desc(rating)) %>%
  select(title, rating) %>%
  head(10) %>%
  knitr::kable()
```

The follow table shows the listing for that user:

title	rating
More (1998)	4.326480
Shawshank Redemption, The (1994)	3.897854
Godfather, The (1972)	3.857806
Aerial, The (La Antena) (2007)	3.834395
Schindler's List (1993)	3.808763
Usual Suspects, The (1995)	3.800428
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	3.774223
Third Man, The (1949)	3.766775
Rear Window (1954)	3.759366
Double Indemnity (1944)	3.757587

Here's the same listing for the Comedy fan:

title	rating
More (1998)	4.159556
Shawshank Redemption, The (1994)	3.730930
Godfather, The (1972)	3.690881
Aerial, The (La Antena) (2007)	3.667471
Schindler's List (1993)	3.641839
Usual Suspects, The (1995)	3.633504
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	3.607299
Third Man, The (1949)	3.599851
Rear Window (1954)	3.592442
Double Indemnity (1944)	3.590663

As expected both lists are identical except for the predicted ratings. Contrast that with the lists produces by the model that includes User/Genre Effects.

Here's the Drama fan:

title	rating
Shawshank Redemption, The (1994)	4.724731
Lives of Others, The (Das Leben der Anderen) (2006)	4.555295
All About Eve (1950)	4.519498
Pather Panchali (1955)	4.509241
12 Angry Men (1957)	4.502257
To Kill a Mockingbird (1962)	4.499709
Word, The (Ordet) (1955)	4.475217
Celebration, The (Festen) (1998)	4.470417
Jean de Florette (1986)	4.464313
Face in the Crowd, A (1957)	4.462676

And here's the Comedy fan:

title	rating
Monty Python and the Holy Grail (1975)	5.119603
Palm Beach Story, The (1942)	5.104120
Tampopo (1985)	5.056567
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	5.054703
I'm Starting From Three (Ricomincio da Tre) (1981)	5.054703
My Man Godfrey (1936)	5.010185
Shall We Dance? (Shall We Dansu?) (1996)	4.974113
Bringing Up Baby (1938)	4.966982
One, Two, Three (1961)	4.942927
Monty Python's And Now for Something Completely Different (1971)	4.939988

Both lists are different and substantially more consistent with the high genre ratings from their respective users.

2.9 Timeline Effects (bt)

Exercises 5-7 in Section 33.8 of the course book describe a method for finding evidence of a time effect by averaging the ratings on a weekly basis. However, there is actually a much stronger time effect that appears at the much shorter time interval of about 8 minutes.

The following graph illustrates this effect by smoothing the mean rating for all 8 minute windows over the period covered by the dataset.

Figure 10: Average Rating for 8 Minute Windows (edx_train)

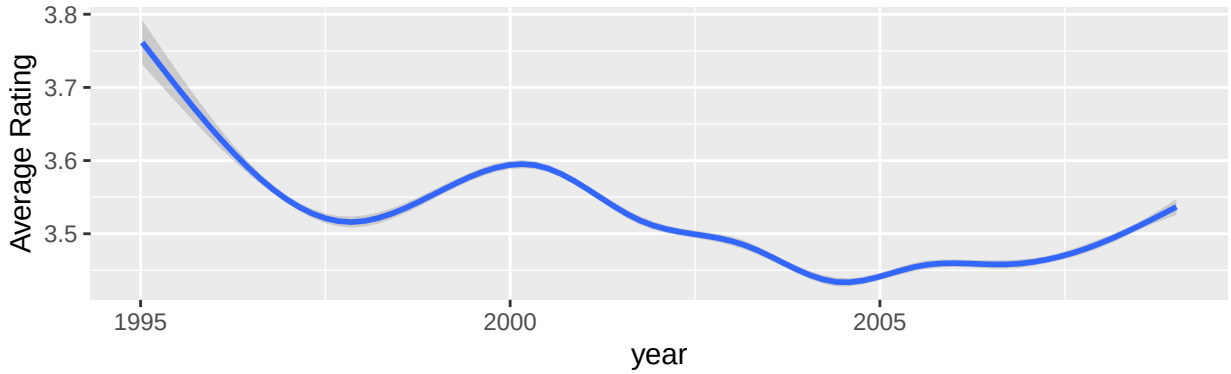
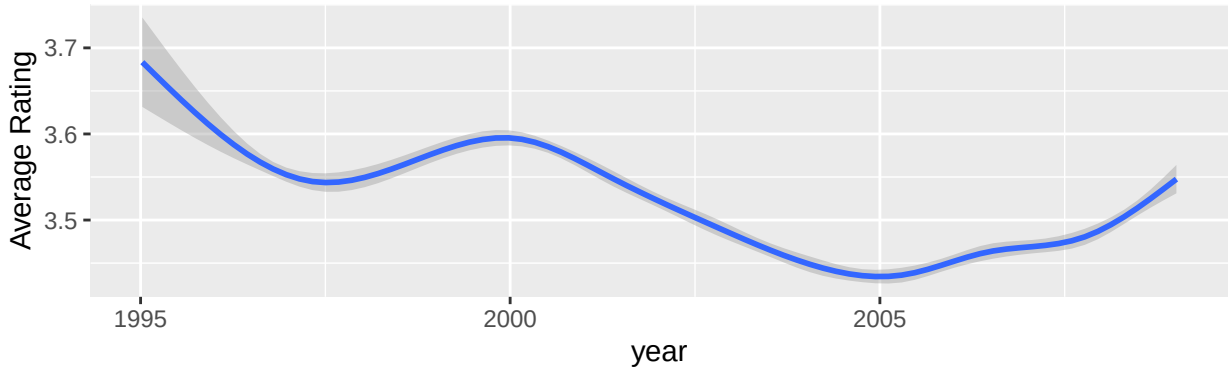


Figure 11: Average Rating for 8 Minute Windows (edx_test)



Notice the graphs are very similar. However, please also note that the point of the graph is not to demonstrate that a smoothing algorithm can be applied as a predictor. The point is more to show that a strong similarity exists between the average ratings for 8 minute time windows in both the edx_train and edx_test datasets.

The actual algorithm is simply to take average value of the remaining residual for an optimized time window and use that average value as the predictor.

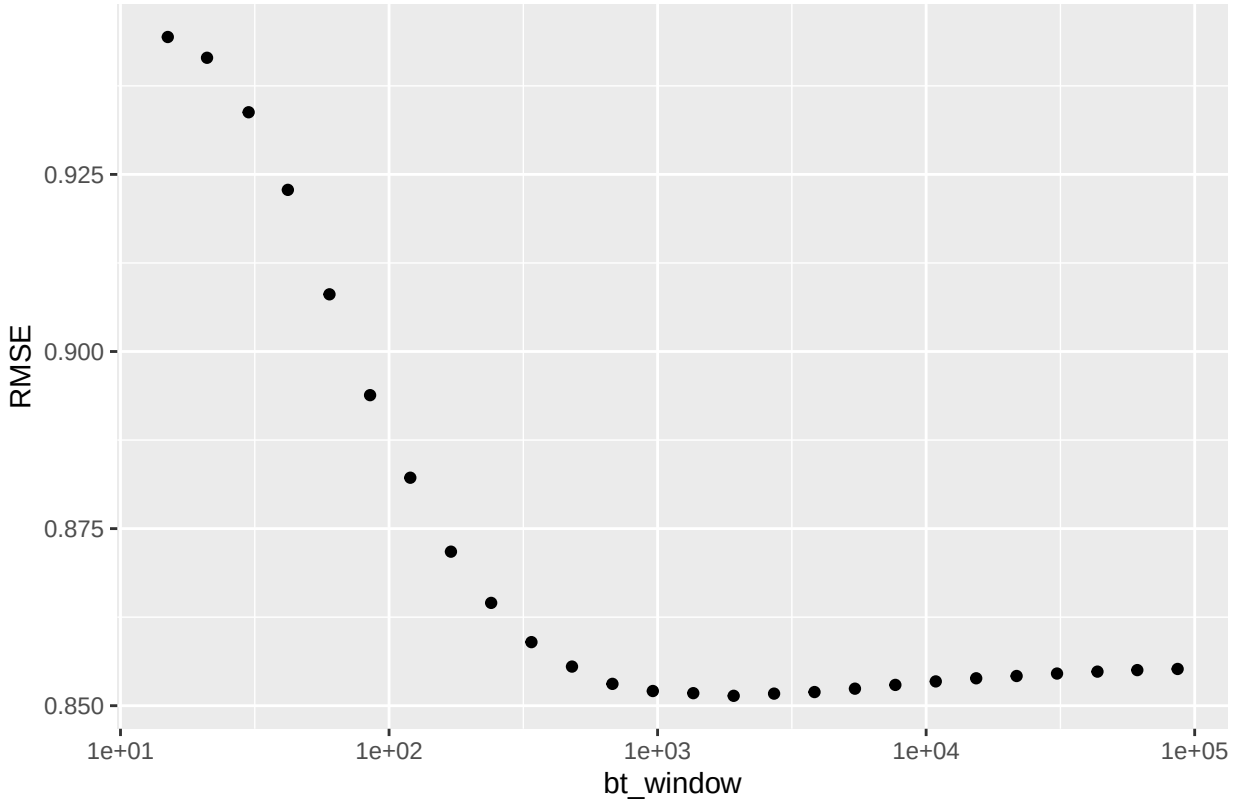
The following code demonstrates how these Timeline Effects are calculated:

```
bt <- edx_train %>%
  left_join(bi_r, by = 'movieId') %>%
  left_join(bu_r, by = 'userId') %>%
  left_join(bg, by = 'genres') %>%
  left_join(bgu_r, by = c('genres', 'userId')) %>%
  mutate(bgu_r = ifelse(is.na(bgu_r), 0, bgu_r)) %>%
  mutate(tsw = timestamp %/% bt_window) %>%
  group_by(tsw) %>%
  summarize(bt = mean(rating - mu - bi_r - bu_r - bg - bgu_r))
y_hat <- mu +
  edx_test %>% left_join(bi_r, by = 'movieId') %>% .$bi_r +
  edx_test %>% left_join(bu_r, by = 'userId') %>% .$bu_r +
  edx_test %>% left_join(bg, by = 'genres') %>% .$bg +
  edx_test %>% left_join(bgu_r, by = c('genres', 'userId')) %>%
  mutate(bgu_r = ifelse(is.na(bgu_r), 0, bgu_r)) %>% .$bgu_r +
  edx_test %>% mutate(tsw = timestamp %/% bt_window) %>%
  left_join(bt, by = 'tsw') %>%
```

```
  mutate(bt = ifelse(is.na(bt), 0, bt)) %>% .$bt
rmse_bt <- rmse(y_hat, edx_test$rating)
```

Note the bt_window variable in the code above. This value is chosen in the same manner as lambdas by iterating through a series of values and selecting the one that results in the lowest RMSE. The following graph shows the results of that iteration process.

Figure 12: RMSE vs Time Window (bt_window) for Timeline Effects



Variable	Calculated Value
bt_window	1920
Timeline Effects RMSE	0.8514

The 8 minute window is from looking at just the ratings values. When looking at the residual values after applying all of the model elements up to this point, that time window exands out to about 30 minutes. The reduction is RMSE is not huge but still significant after all of the previous reductions.

An interpretation of this effect is that ratings that occur close together in time, as in within about 30 minutes, tend to be close enough in value they can be used to predict other ratings made in that same 30 minutes. More research is necessary, but it seems reasonable to speculate that this has something to do with the amount of time that a typical user tends to spend logging into the system and rating sets of movies in a given session.

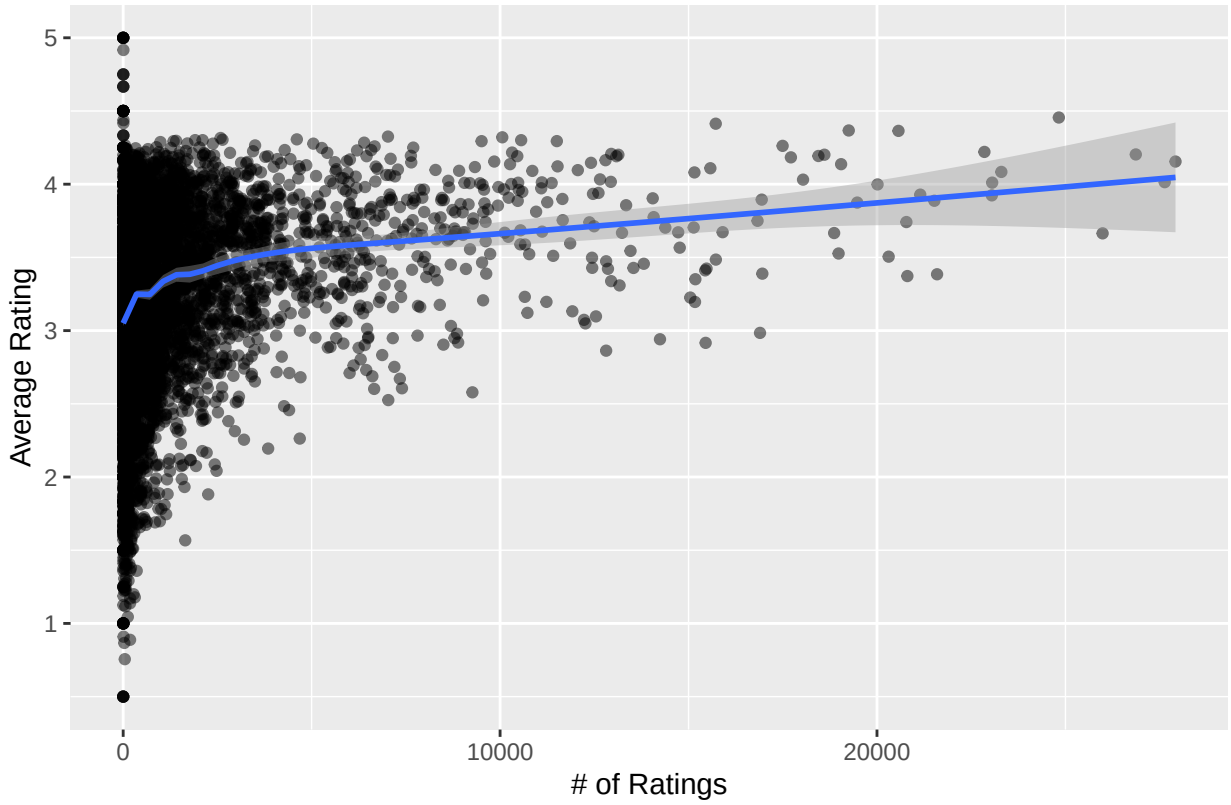
Note also that it's somewhat questionable as to whether this effect would have any practical value in a real movie recommendation system, though that determination would require more research to understand what's really going on here.

2.10 Ratings per Movie Effects

This section explores Ratings per Movie Effects or the tendency for movies with fewer ratings to have lower average ratings.

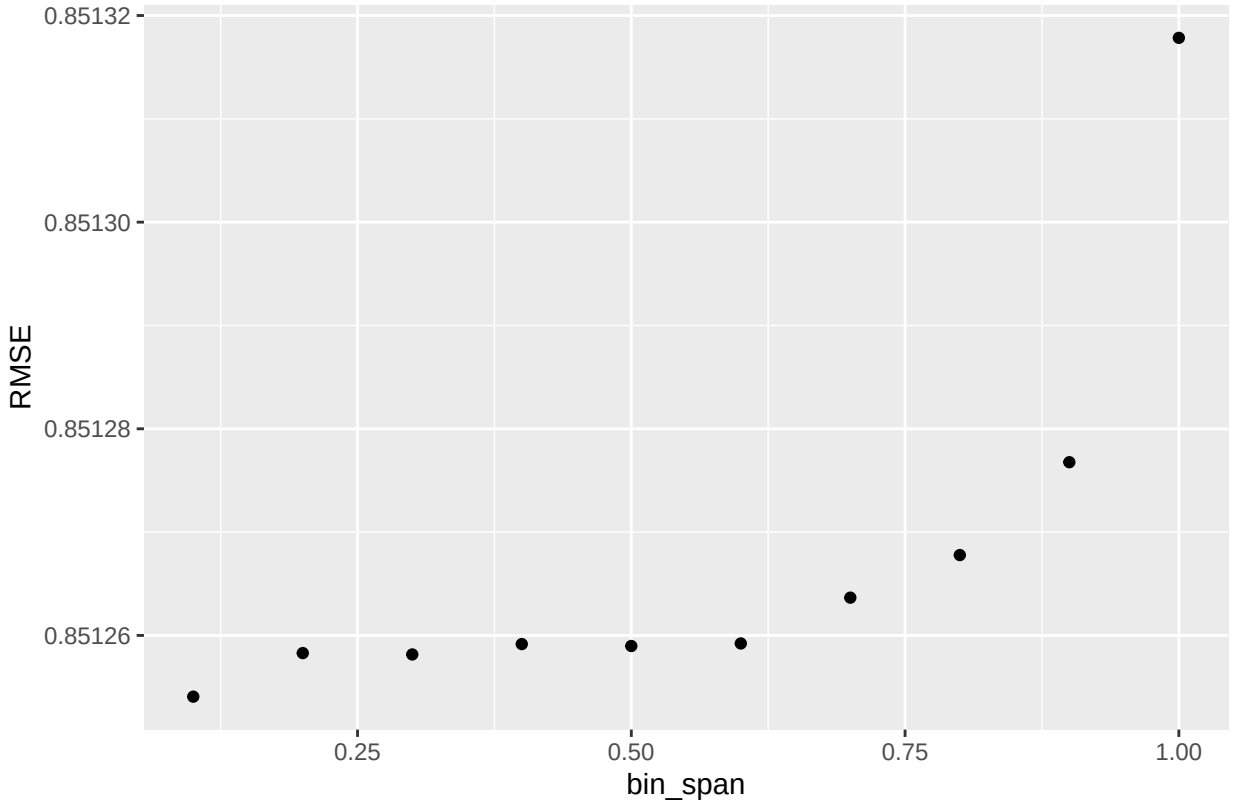
The following chart illustrates the tendency by displaying the average movie rating vs. the number of ratings for the Capstone dataset.

Figure 13: Average Rating vs Number of Ratings per Movie



Similar to the discussion for Genre Effects, however, the expectation is this effect would largely disappear once the average movie rating per movie is added to the model as it is in Section 2.5. The attempt to capture the effect with LOESS smoothing and optimize for the span parameter is shown in the following graph. The results using the optimal span value are shown immediately after.

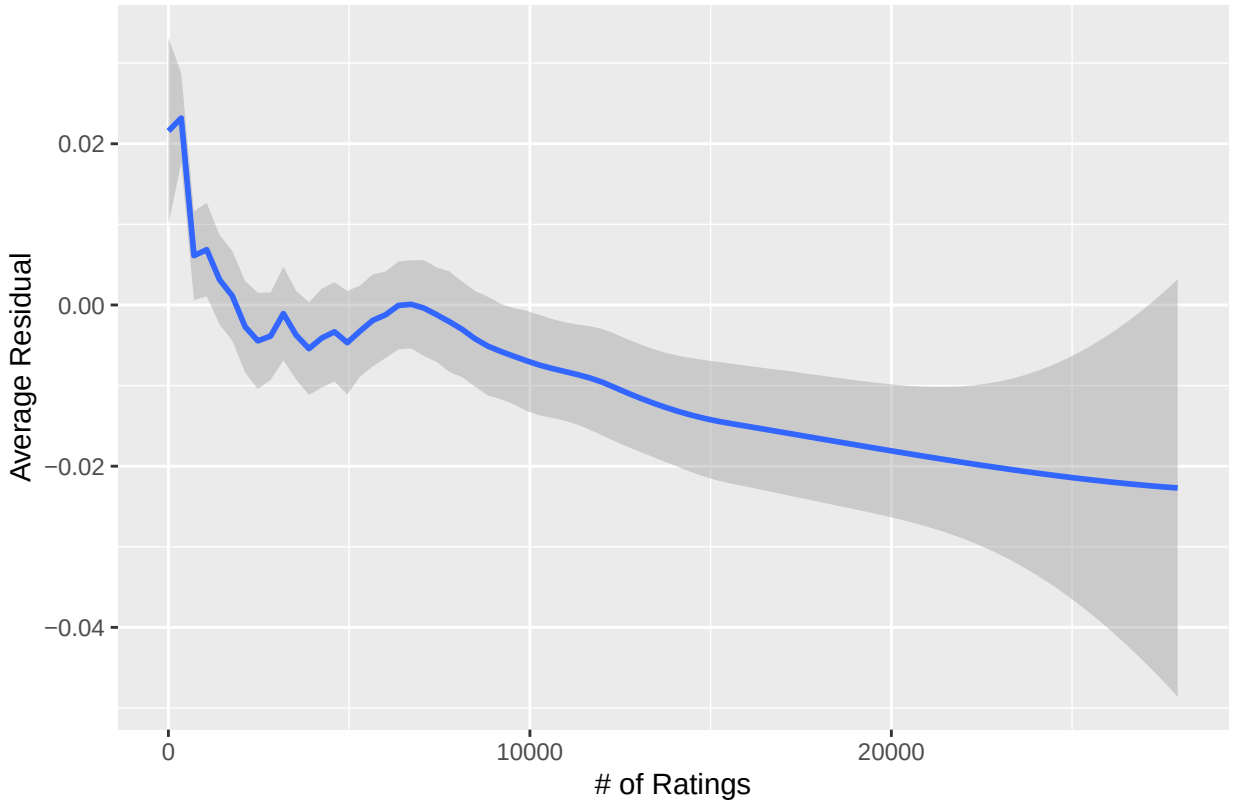
Figure 14: RMSE vs Span (bin_span) for Ratings per Movie Effects



Variable	Calculated Value
bin_span	0.1
Ratings per Movie Effects RMSE	0.85125

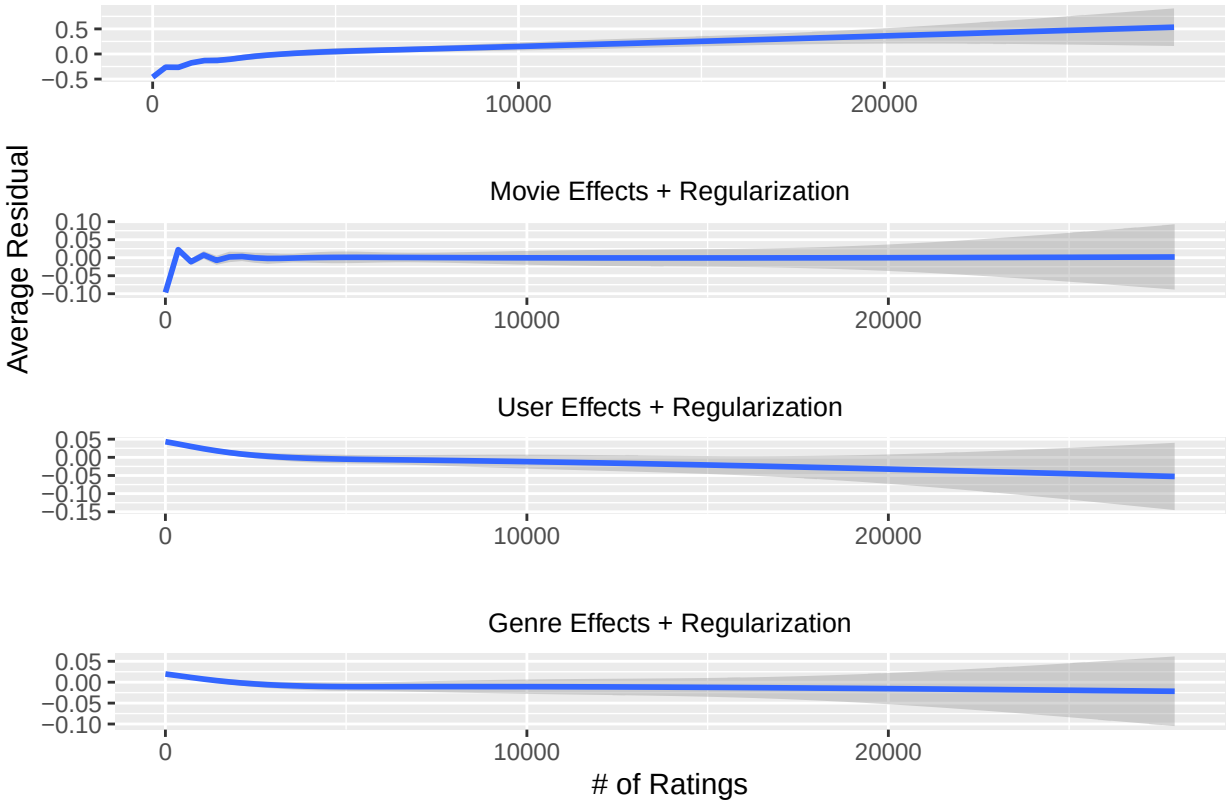
As anticipated, the effect is quite minimal. The following chart depects the LOESS smoothing when applied to the model average residual values instead of the original movie ratings.

Figure 15: Average Residual vs Number of Ratings per Movie



Somewhat surprisingly the orientation of the effect appears to have flipped so that movies with lower numbers of ratings tend to have higher residual values. To better understand why that happened, the following chart illustrates the average residual vs. number of ratings at various stages of model development.

Figure 16: Average Residual at Various Stages



From the chart it appears that Movie Effects essentially zeroes out the Number of Ratings Effect and User Effects brings it back but with a reversed orientation.

2.11 Summary of Training Results

With that the discussion of the different model components is complete. The follow tables summarize the training results including the optimal values for tuning parameters and the observed RMSE values for each step.

2.11.1 Tuning Parameters

Parameter	Value
lmv	2.5
lu	5
lg	0
lgu	8
bt_window	1920
bin_span	0.1

2.11.2 RMSE

Description	RMSE
Random Guess	1.94172
Average Rating (mu)	1.06
Movie Effects (bi)	0.94311

Description	RMSE
Movie Effects w/ Regularization (bi_r)	0.94306
User Effects (bu)	0.86495
User Effects w/ Regularization (bu_r)	0.86451
Genre Effects (bg)	0.86421
User/Genre Effects (bgu)	0.91847
User/Genre Effects w/ Regularization (bgu_r)	0.85572
Timeline Effects (bt)	0.8514
Ratings per Movie Effects (bin)	0.85125

2.12 Cross Validation Sanity Check

As mentioned above Section 2.2 the models were to be run again with different seed values as a sanity check. The results are displayed below

Description	Seed = 1	Seed = 2	Seed = 3	Seed = 2*	Seed = 3*
Training Results	0.85125	0.85221	0.85211	0.85220	0.85210

* With Seed 1 tuning parameters.

The differences in results are in the 0.1% range which is small but not nothing. However, more importantly when running Seed 2 and Seed 3 with the Seed 1 parameters the change for each Seed from their own parameters is 0.00001 in both cases which seems to suggest that doing full k-fold Cross Validation with $k > 1$ is unnecessary.

For completeness, the following table lists the tuning parameters for each seed.

Parameter	Seed = 1	Seed = 2	Seed = 3
lmv	2.5	1.75	2
lu	5	5.25	4.75
lg	0	10	10
lgu	8	8	8
bt_window	1920	1920	1920
bin_span	0.1	0.1	0.1

Nothing really surprising except the 10's for lg. But those values are effectively multiplied by zero so they're not significant.

3 Results

The following code was used in combination with the tuning parameters listed in Section 2.1.11 to calculate the final value for RMSE which is listed immediately afterwards:

```
# Calculate the final results with edx and final_holdout_test.
mu_final <- mean(edx$rating)

bi_r_final <- edx %>%
  group_by(movieId) %>%
  summarize(bi_r_final = sum(rating - mu_final)/(n() + lmv))

bu_r_final <- edx %>%
```

```
  left_join(bi_r_final, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(bu_r_final = sum(rating - mu_final - bi_r_final)/(n() + lu))

bg_final <- edx %>%
  left_join(bi_r_final, by = 'movieId') %>%
  left_join(bu_r_final, by = 'userId') %>%
  group_by(genres) %>%
  summarize(bg_final = mean(rating - mu_final - bi_r_final - bu_r_final))

bgu_r_final <- edx %>%
  left_join(bi_r_final, by = 'movieId') %>%
  left_join(bu_r_final, by = 'userId') %>%
  left_join(bg_final, by = 'genres') %>%
  group_by(genres, userId) %>%
  summarize(bgu_r_final = sum(rating - mu_final - bi_r_final - bu_r_final -
                              bg_final)/(n() + lgu))

bt_final <- edx %>%
  left_join(bi_r_final, by = 'movieId') %>%
  left_join(bu_r_final, by = 'userId') %>%
  left_join(bg_final, by = 'genres') %>%
  left_join(bgu_r_final, by = c('genres', 'userId')) %>%
  mutate(bgu_r_final = ifelse(is.na(bgu_r_final), 0, bgu_r_final)) %>%
  mutate(tsw = timestamp %/% bt_window) %>%
  group_by(tsw) %>%
  summarize(bt_final = mean(rating - mu_final - bi_r_final - bu_r_final -
                            bg_final - bgu_r_final))

bin_data_final <- edx %>%
  left_join(bi_r_final, by = 'movieId') %>%
  left_join(bu_r_final, by = 'userId') %>%
  left_join(bg_final, by = 'genres') %>%
  left_join(bgu_r_final, by = c('genres', 'userId')) %>%
  mutate(bgu_r_final = ifelse(is.na(bgu_r_final), 0, bgu_r_final)) %>%
  mutate(tsw = timestamp %/% bt_window) %>%
  left_join(bt_final, by = 'tsw') %>%
  mutate(bt_final = ifelse(is.na(bt_final), 0, bt_final)) %>%
  group_by(movieId) %>%
  mutate(n = n()) %>% ungroup() %>%
  group_by(n) %>%
  summarize(avg_bias = mean(rating - mu_final - bi_r_final - bu_r_final -
                            bg_final - bgu_r_final - bt_final))

bin_fit_final <- loess(avg_bias~n, data=bin_data_final, span=bin_span)
bin_final <- edx %>%
  group_by(movieId) %>%
  summarize(n = n()) %>%
  mutate(bin_final = predict(bin_fit_final, .)) %>%
  select(movieId, bin_final)

y_hat <- mu_final +
  final_holdout_test %>% left_join(bi_r_final, by = 'movieId') %>% .$bi_r_final +
```

```
  final_holdout_test %>% left_join(bu_r_final, by = 'userId') %>% .$bu_r_final +
  final_holdout_test %>% left_join(bg_final, by = 'genres') %>% .$bg_final +
  final_holdout_test %>% left_join(bgu_r_final, by = c('genres', 'userId')) %>%
  mutate(bgu_r_final = ifelse(is.na(bgu_r_final), 0, bgu_r_final)) %>% .$bgu_r_final +
  final_holdout_test %>% mutate(tsw = timestamp %/% bt_window) %>%
  left_join(bt_final, by = 'tsw') %>%
  mutate(bt_final = ifelse(is.na(bt_final), 0, bt_final)) %>% .$bt_final +
  final_holdout_test %>% left_join(bin_final, by = 'movieId') %>% .$bin_final

rmse_final <- rmse(y_hat, final_holdout_test$rating)
```

The following table includes the final RMSE value observed when calibrating the model with full edx training set and running it against the final_holdout test set. Note that only the values that can be calculated using only the edx dataset are recalculated. These include mu, bi_r, bu_r, bg, bgu_r, bt, and bin. None of the tuning parameters are recalculated as doing so would require using the final_holdout_test set which is not allowed.

Description	Value
Final Result	0.85026

Somewhat surprisingly the model performs better against the final holdout test than with the training set. However, at least some of this may have to do with the full final sets being about 10% larger than the training sets. A similar effect was observed in the course material with the much smaller datasets not performing as well as the Capstone training sets in Sections 2.5 and 2.6 above.

4 Conclusion

Utilizing the strategies from this project, it's possible to achieve a better RMSE than what was required to win the Netflix Prize (0.8572). Granted, the genre data elements present in the MovieLens dataset that were required to get below the threshold were not present in the Neflix dataset so it's probably not the quite accomplishment that it sounds.

What's more it's clear from sections 2.5 and 2.8 that minimizing RMSE is not the whole story. It's likely that most end users care more about rating order than about rating magnitude. Only Section 2.8 adds a model element that can change the rating order for different users.

While there are may additional effects that could be explored as additions to the model that would further reduce RMSE, it might be more worthwhile to look at algorithms, such as nearest neighbor, that might seem to have a better chance at providing user specific results.

5 References

- Data Science: Machine Learning, https://www.edx.org/course/data-science-machine-learning
- Recommendation Systems, http://rafalab.dfci.harvard.edu/dsbook/large-datasets.html#recommendation-systems
- Regularization, http://rafalab.dfci.harvard.edu/dsbook/large-datasets.html#regularization
- Netflix Awards $1 Million Prize and Starts a New Contest, https://archive.nytimes.com/bits.blogs.nytimes.com/2009/09/21/netflix-awards-1-million-prize-and-starts-a-new-contest/
- The Netflix Prize: How a $1 Million Contest Changed Binge-Watching Forever, https://www.thrillist.com/entertainment/nation/the-netflix-prize

- Netflix Never Used Its $1 Million Algorithm Due To Engineering Costs, https://www.wired.com/2012/04/netflix-prize-costs/
- Winning the Netflix Prize: A Summary, http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/
- MovieLens 10M Dataset, https://grouplens.org/datasets/movielens/10m/
- MovieLens Datasets: Context and History, https://grouplens.org/blog/movielens-datasets-context-and-history