Harvard Data Science Capstone

Ekundayo Azubuike

2024-11-21

Introduction

Data Overview

The following analysis and modeling center on the `movielens` data set (you can read more about it here). This `data.frame` is a subset of a larger data set containing 9,000,055 rows of observations and six columns of variables: `userId`, `movieId`, `rating`, `timestamp`, `title`, and `genres`. The target variable is `rating`, and the other five variables are predictors.

Purpose

The goal of the analysis that follows is to build and train a statistical model to predict the `rating` for each row in the holdout set given a set of predictor variables. Root mean squared error (RMSE) will be the metric used to evaluate the quality of the model.

Procedure

1. First, I set up my environment by creating a new R project and populating the required files: `capstone_report.Rmd`, `capstone_report.pdf`, and `capstone_code.R`.
2. Next, I loaded the `movielens` data according to the course instructions.
3. I then performed some pre-processing to change a few data types, add new predictors, and split the data into training and test sets.
4. I used the pre-processed data to perform exploratory data analysis to develop a preliminary understanding of the data.
5. I performed variable selection and built a linear model of `rating` as a function of a subset of the original predictor variables: `movieId`, `userId`, `genres`, and `year`.
6. I calculated regularized bias terms for each predictor.
7. I calculated RMSE on the final model.
8. Finally, I tested the model against the holdout set, yielding a final RMSE of less than `0.86549`.

Methods/Analysis

Loading Course Code

The following analysis will leverage functionality from the `tidyverse` library for data cleaning, manipulation, and visualization. I used `caret` for pre-processing and `Hmisc` for some exploratory data analysis.

```r
# install and load relevant libraries
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(Hmisc)) install.packages("Hmisc", repos = "http://cran.us.r-project.org")
if(!require(dslabs)) install.packages("dlabs", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)
library(Hmisc)
library(dslabs)
library(knitr)
ds_theme_set()

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

options(timeout = 120)

dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)

movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

movielens <- left_join(ratings, movies, by = "movieId")

# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
```

```r
temp <- movielens[test_index,]

# Make sure userId and movieId in final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Inspecting Data

The data is contained within a `data.frame` object, and it contains three integer vectors (`userId`, `movieId`, and `timestamp`), two character vectors (`title` and `genres`), and a target vector of doubles (`rating`). The `summary()` doesn't provide a very useful overview because many of the variables are not properly encoded, but we do get a somewhat useful understanding of the distribution of the target variable `rating` given the measures of centrality. The mean and median values for `rating` are `3.512` and `4.000` respectively.

```r
# view summary of edx data set
str(edx)
```

```
## 'data.frame':    9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : int  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 8
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|A
```

```r
class(edx)
```

```
## [1] "data.frame"
```

```r
summary(edx) %>% kable()
```

	userId	movieId	rating	timestamp	title	genres
	Min. : 1	Min. : 1	Min. :0.500	Min. :7.897e+08	Length:9000055	Length:9000055
	1st Qu.:18124	1st Qu.: 648	1st Qu.:3.000	1st Qu.:9.468e+08	Class :character	Class :character
	Median :35738	Median : 1834	Median :4.000	Median :1.035e+09	Mode :character	Mode :character
	Mean :35870	Mean : 4122	Mean :3.512	Mean :1.033e+09	NA	NA
	3rd Qu.:53607	3rd Qu.: 3626	3rd Qu.:4.000	3rd Qu.:1.127e+09	NA	NA

userId	movieId	rating	timestamp	title	genres
Max. :71567	Max. :65133	Max. :5.000	Max. :1.231e+09	NA	NA

```
head(edx) %>% kable()
```

	userId	movieId	rating	timestamp	title	genres
1	1	122	5	838985046	Boomerang (1992)	Comedy|Romance
2	1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
4	1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
5	1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
6	1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi
7	1	355	5	838984474	Flintstones, The (1994)	Children|Comedy|Fantasy

Data Pre-processing

As a first step, I created a `year` column by using a regular expression (`"\\((\\d{4})\\)$"`) to match the characters between the parentheses at the end of the `title` string and converting the match to a factor. Similarly, I converted the `userId`, `movieId`, and `genres` columns to factors as well for analysis. For the `genres` column, I also extracted the primary categorization with a regular expression (`"^[^|]+"`) to reduce the number of levels in the factor for analysis. Lastly, I converted the `timestamp` column to a datetime and reviewed the summary with the recoded values. Already, it's clear that some users have supplied more ratings than others. Moreover, some movies receive more ratings than others. `Action`, `Comedy`, and `Drama` are the genres with the greatest number of reviews. Lastly, different years have different numbers of reviews, with `1995` having the greatest number. All of this information points to the idea that these four variables (`userId`, `movieId`, `genres`, and `year`) may benefit from regularization to account for the differences in numbers of ratings per category. The `date` column ranges from `29 January 1996` to `5 January 2009`.

```
# recode variables and view summary
edx.mutated <- edx %>%
  mutate(year = factor(str_match(edx$title, "\\((\\d{4})\\)$")[,2]),
         date = as_datetime(timestamp),
         userId = factor(userId),
         movieId = factor(movieId),
         genres = factor(str_match(edx$genres, "^[^|]+")))

summary(edx.mutated) %>% kable()
```

userId	movieId	rating	timestamp	title	genres	year	date
59269 : 6616	296 : 31362	Min. :0.500	Min. :7.897e+08	Length:9000055	Action :2560545	1995 : 786762	Min. :1995-01-09 11:46:49.00
67385 : 6360	356 : 31079	1st Qu.:3.000	1st Qu.:9.468e+08	Class :character	Comedy :2437260	1994 : 671376	1st Qu.:2000-01-01 23:11:23.00
14463 : 4648	593 : 30382	Median :4.000	Median :1.035e+09	Mode :character	Drama :1741668	1996 : 593518	Median :2002-10-24 21:11:58.00

userId	movieId	rating	timestamp	title	genres	year	date
68259 : 4036	480 : 29360	Mean :3.512	Mean :1.033e+09	NA	Adventure: 753650	1999 : 489537	Mean :2002-09-21 13:45:07.38
27468 : 4023	318 : 28015	3rd Qu.:4.000	3rd Qu.:1.127e+09	NA	Crime : 529521	1993 : 481184	3rd Qu.:2005-09-15 02:21:21.00
19635 : 3771	110 : 26212	Max. :5.000	Max. :1.231e+09	NA	Horror : 233074	1997 : 429751	Max. :2009-01-05 05:02:16.00
(Other):8970601	(Other):8823645	NA	NA	NA	(Other) : 744337	(Other):5547027	NA

```r
head(edx.mutated) %>% kable()
```

	userId	movieId	rating	timestamp	title	genres	year	date
1	1	122	5	838985046	Boomerang (1992)	Comedy	1992	1996-08-02 11:24:06
2	1	185	5	838983525	Net, The (1995)	Action	1995	1996-08-02 10:58:45
4	1	292	5	838983421	Outbreak (1995)	Action	1995	1996-08-02 10:57:01
5	1	316	5	838983392	Stargate (1994)	Action	1994	1996-08-02 10:56:32
6	1	329	5	838983392	Star Trek: Generations (1994)	Action	1994	1996-08-02 10:56:32
7	1	355	5	838984474	Flintstones, The (1994)	Children	1994	1996-08-02 11:14:34

Training and Testing Subsets

I created a 20% partition on the target variable `rating` to generate a testing index. I used that index to subset the original data so that I could trian my model on 80% of the original data and test it on the remaining 20%. I used the `semi_join()` to ensure that all movies represented in the training set were also present in the testing set.

```r
# partition testing and training sets
test.index <- createDataPartition(edx.mutated$rating,
                                  p = 0.2,
                                  list = F)
edx.test <- edx.mutated[test.index, ]
edx.train <- edx.mutated[-test.index, ]

# ensure that all movies in training set are also in testing set
edx.test <- edx.test %>%
  semi_join(edx.train, by = "userId")
edx.train <- edx.train %>%
  semi_join(edx.test, by = "userId")
```

Exploratory Data Analysis

Predictor Correlation

I began my exploratory data analysis by computing a Spearman correlation matrix to determine if any of the variables were too highly correlated. The highest correlation (`0.50`) exists between the `movieId` and `timestamp` variables, but the correlation isn't very strong, so it shouldn't affect analysis.

```
# correlation matrix
edx.train %>%
  select(-c(rating, genres, title, date)) %>%
  as.matrix() %>%
  Hmisc::rcorr(type = "spearman")
```

```
##           userId movieId timestamp year
## userId      1.00    0.01      0.02 0.00
## movieId     0.01    1.00      0.50 0.33
## timestamp   0.02    0.50      1.00 0.22
## year        0.00    0.33      0.22 1.00
##
## n= 7197526
##
##
## P
##           userId movieId timestamp year
## userId           0.000   0.000     0.662
## movieId   0.000          0.000     0.000
## timestamp 0.000  0.000             0.000
## year      0.662  0.000   0.000
```

Stratification, Summary, and Visualization

Next, I stratified the data by each categorical predictor (`movieId`, `userId`, `genres`, and `year`) in order to uncover patterns in the distribution of rating counts and measures of centrality of the rating (mean and standard deviation).

Stratification by `movieId` As there is a large range of number of reviews (`25114`), I am justified in applying regularization to the `movieId` bias term later on in the analysis to lessen the impact of movies with fewer reviews on the predictions.

```
# examine the distribution of ratings by movie
edx.train %>%
  group_by(movieId) %>%
  summarise(movie.avg = mean(rating),
            movie.sd = sd(rating),
            rating.count = n()) %>%
  arrange(desc(rating.count)) %>%
  slice(1:10) %>%
  kable()
```

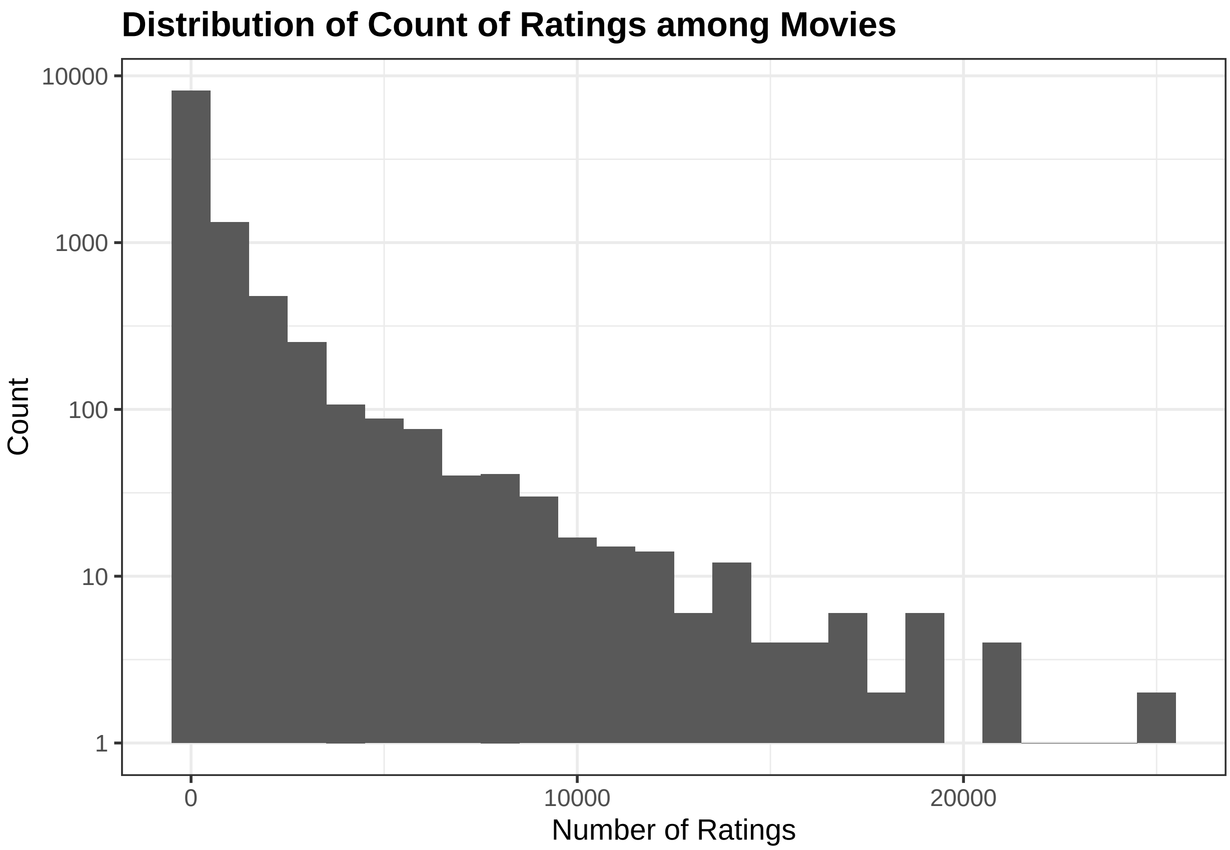
movieId	movie.avg	movie.sd	rating.count
296	4.155604	1.0012415	25115
356	4.011619	0.9708657	24916
593	4.202483	0.8405775	24402
480	3.663599	0.9378060	23472
318	4.457967	0.7161132	22387
110	4.083158	0.9527615	20870
457	4.010198	0.7773298	20838
589	3.926932	0.9091842	20741
260	4.222360	0.9118768	20577
150	3.888378	0.8519416	19472

```
edx.train %>%
  group_by(movieId) %>%
  summarise(movie.avg = mean(rating),
            rating.count = n()) %>%
  summarise(range = max(rating.count) - min(rating.count)) %>%
  kable()
```

range
25114

A histogram of the distribution of rating counts among movies further illustrates the point above: it is more common for movies to have 2,000 or fewer reviews. The distribution is not normal; it has positive skew (that is, its right tail is longer due to the higher prevalence of smaller values).

```
# histogram of the distribution of rating counts among movies
edx.train %>%
  group_by(movieId) %>%
  summarise(movie.avg = mean(rating),
            rating.count = n()) %>%
  ggplot(aes(x = rating.count)) +
  geom_histogram(binwidth = 1000) +
  scale_y_log10() +
  ggtitle("Distribution of Count of Ratings among Movies") +
  xlab("Number of Ratings") +
  ylab("Count")
```



Stratification by `userId` Stratification by `userId` reveals a similar but less dramatic range in rating counts between users (`5317`) that justifies a regularization procedure on the `userId` bias term.

```
# examine the distribution of ratings by user
edx.train %>%
  group_by(movieId) %>%
  summarise(user.avg = mean(rating),
            user.sd = sd(rating),
            rating.count = n()) %>%
  arrange(desc(rating.count)) %>%
  slice(1:10) %>%
  kable()
```

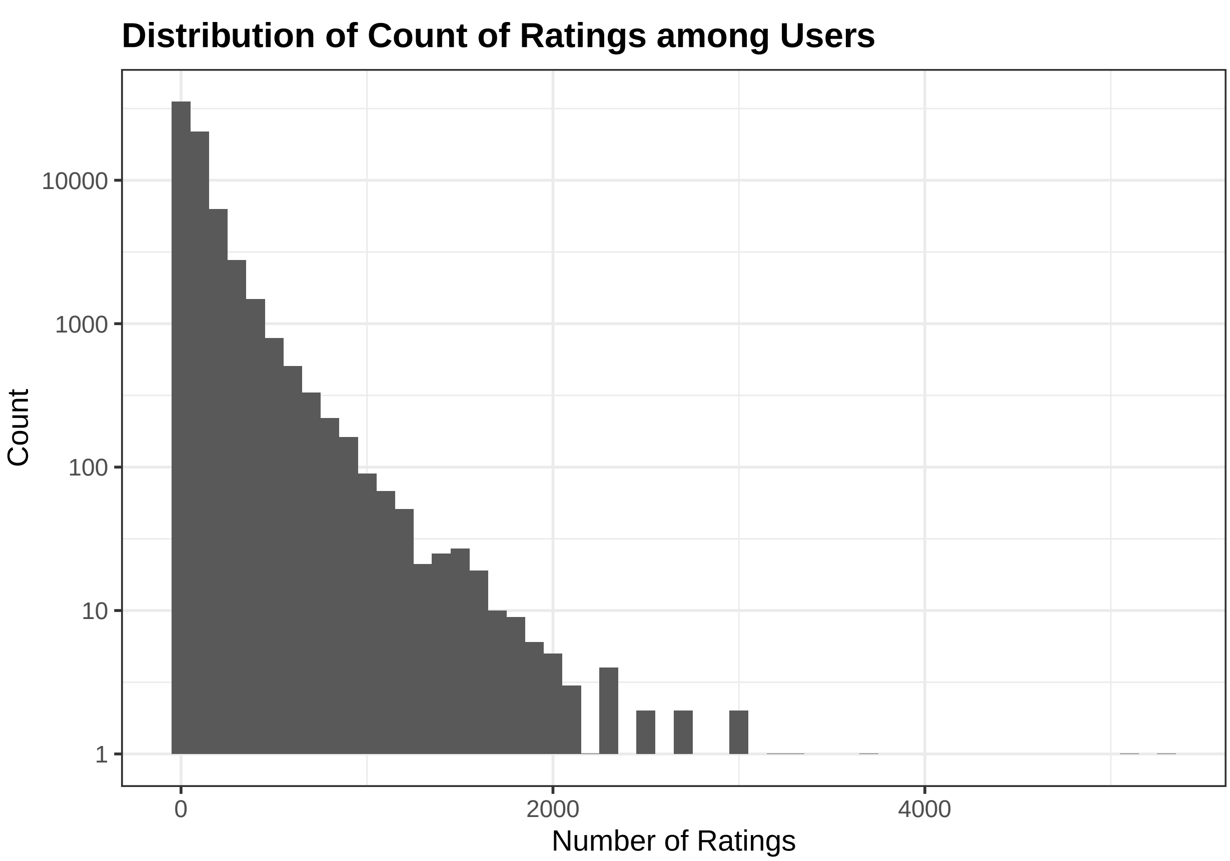
movieId	user.avg	user.sd	rating.count
296	4.155604	1.0012415	25115
356	4.011619	0.9708657	24916
593	4.202483	0.8405775	24402
480	3.663599	0.9378060	23472
318	4.457967	0.7161132	22387
110	4.083158	0.9527615	20870
457	4.010198	0.7773298	20838
589	3.926932	0.9091842	20741
260	4.222360	0.9118768	20577
150	3.888378	0.8519416	19472

```r
edx.train %>%
  group_by(userId) %>%
  summarise(user.avg = mean(rating),
            rating.count = n()) %>%
  summarise(range = max(rating.count) - min(rating.count)) %>%
  kable()
```

range
5317

The histogram of rating counts among users also has positive skew, indicating that smaller values predominate.

```r
# histogram of the distribution of rating counts among users
edx.train %>%
  group_by(userId) %>%
  summarise(user.avg = mean(rating),
            ratings.count = n()) %>%
  ggplot(aes(x = ratings.count)) +
  geom_histogram(binwidth = 100) +
  scale_y_log10() +
    ggtitle("Distribution of Count of Ratings among Users") +
  xlab("Number of Ratings") +
  ylab("Count")
```



Stratification by `genres` Stratification by `genres` indicates that `Action` movies are the most frequently rated, whereas `Film-Noir` movies receive the highest rating on average. A large range of rating counts between genres (`2047641`) justifies a regularization procedure on the `genres` bias term.

```r
# counts, average ratings, and range by genre
edx.train %>%
  group_by(genres) %>%
  summarise(genre.avg = mean(rating),
            count = n()) %>%
  arrange(desc(count)) %>%
  kable()
```

genres	genre.avg	count
Action	3.421619	2047647
Comedy	3.453687	1949021
Drama	3.666679	1392410
Adventure	3.564667	602730
Crime	3.871234	423772
Horror	3.032229	186649
Animation	3.550586	174536
Children	3.246862	144745
Thriller	3.531206	75754
Documentary	3.788425	64776
Sci-Fi	3.430775	40101
Mystery	3.699514	24472
Fantasy	3.313265	20821
Musical	3.637403	13031
Film-Noir	4.142583	12754
Western	3.533891	12260
Romance	3.283274	10188
War	3.684239	1840
IMAX	2.230769	13
(no genres listed)	3.666667	6

```r
edx.train %>%
  group_by(genres) %>%
  summarise(genre.avg = mean(rating),
            count = n()) %>%
  arrange(desc(genre.avg)) %>%
  kable()
```

genres	genre.avg	count
Film-Noir	4.142583	12754
Crime	3.871234	423772
Documentary	3.788425	64776
Mystery	3.699514	24472
War	3.684239	1840
Drama	3.666679	1392410
(no genres listed)	3.666667	6
Musical	3.637403	13031

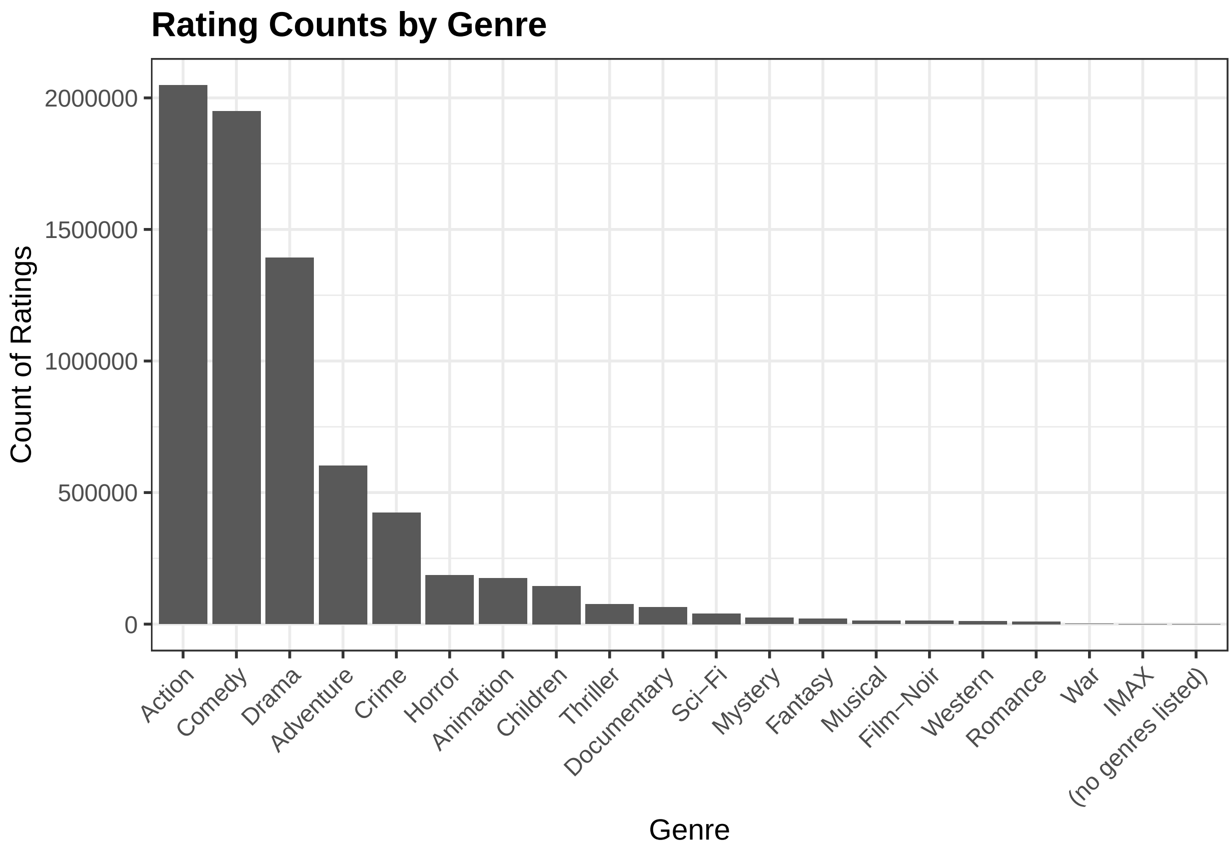
genres	genre.avg	count
Adventure	3.564667	602730
Animation	3.550586	174536
Western	3.533891	12260
Thriller	3.531206	75754
Comedy	3.453687	1949021
Sci-Fi	3.430775	40101
Action	3.421619	2047647
Fantasy	3.313265	20821
Romance	3.283274	10188
Children	3.246862	144745
Horror	3.032229	186649
IMAX	2.230769	13

```
edx.train %>%
  group_by(genres) %>%
  summarise(genre.avg = mean(rating),
            rating.count = n()) %>%
  summarise(range = max(rating.count) - min(rating.count)) %>%
  kable()
```

range
2047641

A bar graph illustrates the stark differences in rating counts between genres.

```
# bar graph of rating counts by genre
edx.train %>%
  group_by(genres) %>%
  summarise(genre.avg = mean(rating),
            rating.count = n()) %>%
  ggplot(aes(x = reorder(genres, desc(rating.count)), y = rating.count)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1)) +
  labs(title = "Rating Counts by Genre") +
  xlab("Genre") +
  ylab("Count of Ratings")
```



Stratification by `year` Stratification by `year` highlights two points. First, movies that were released in 1946 have the highest average rating. Second, movies released in the 1990's consistently have the highest number of ratings. The range in number of ratings between the most- and least-rated years is `629022`, justifying a regularization procedure on the `year` bias term.

```
# counts, average ratings, and range by year
edx.train %>%
  group_by(year) %>%
  summarise(year.avg = mean(rating),
            rating.count = n()) %>%
  arrange(desc(year.avg)) %>%
  head(20) %>%
  kable()
```

year	year.avg	rating.count
1934	4.054029	4766
1946	4.053627	13538
1942	4.046771	16089
1941	4.020090	19263
1931	4.018929	6181
1957	4.012780	19522
1954	4.006534	24104
1927	4.000604	3310
1962	3.990199	26834
1944	3.990120	9615

year	year.avg	rating.count
1952	3.984296	9265
1936	3.963973	3317
1972	3.954009	31658
1935	3.947997	4942
1949	3.944507	6226
1948	3.943099	7926
1951	3.942791	17366
1924	3.942667	375
1938	3.941675	6258
1920	3.930851	470

```
edx.train %>%
  group_by(year) %>%
  summarise(year.avg = mean(rating),
            rating.count = n()) %>%
  arrange(desc(year.avg)) %>%
  tail(20) %>%
  kable()
```

year	year.avg	rating.count
1994	3.472427	536780
1985	3.469999	111846
1989	3.466854	182737
1988	3.465090	137239
1986	3.461705	140594
1978	3.460758	43830
2003	3.458629	169104
2008	3.454059	21386
1999	3.452428	391646
2002	3.451468	217277
1995	3.442592	629047
2001	3.439443	244347
1992	3.431593	189228
1998	3.416500	321905
1990	3.397056	184750
2000	3.394694	306213
1997	3.365133	343906
1915	3.363636	143
1996	3.362030	474475
1919	3.339130	115

```
edx.train %>%
  group_by(year) %>%
  summarise(year.avg = mean(rating),
            rating.count = n()) %>%
  arrange(desc(rating.count)) %>%
  slice(1:20) %>%
  kable()
```

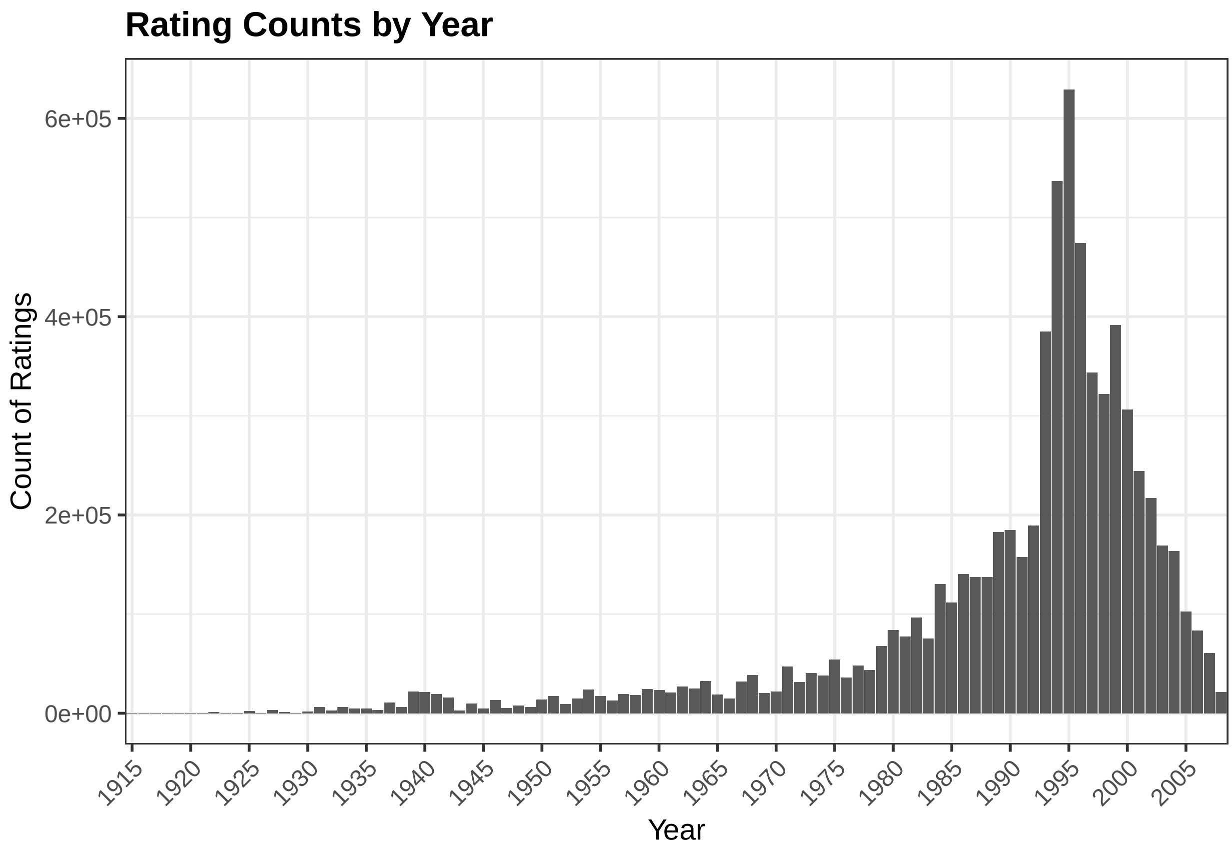
year	year.avg	rating.count
1995	3.442592	629047
1994	3.472427	536780
1996	3.362030	474475
1999	3.452428	391646
1993	3.496240	384853
1997	3.365133	343906
1998	3.416500	321905
2000	3.394694	306213
2001	3.439443	244347
2002	3.451468	217277
1992	3.431593	189228
1990	3.397056	184750
1989	3.466854	182737
2003	3.458629	169104
2004	3.530170	163521
1991	3.572297	157427
1986	3.461705	140594
1987	3.505349	137505
1988	3.465090	137239
1984	3.558070	130480

```
edx.train %>%
  group_by(year) %>%
  summarise(year.avg = mean(rating),
            rating.count = n()) %>%
  summarise(range = max(rating.count) - min(rating.count)) %>%
  kable()
```

range
629022

A bar graph of rating counts by year clearly illustrates the higher frequency of ratings among movies released in the 1990's.

```
# bar graph of rating counts by year
edx.train %>%
  group_by(year) %>%
  summarise(year.avg = mean(rating),
            rating.count = n()) %>%
  ggplot(aes(x = year, y = rating.count)) +
  geom_bar(stat = "identity") +
  scale_x_discrete(breaks = seq(1915, 2008, 5)) +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1)) +
  labs(title = "Rating Counts by Year") +
  xlab("Year") +
  ylab("Count of Ratings")
```



Bias Term Regularization

Setup and Overview

The next step was to calculate regularized bias terms for the selected predictors (`movieId`, `userId`, `genres`, and `year`). I first stored the overall mean (`mu`) and test ranges for `lambda` in variables. I ultimately ran my code using the variable `even.more.lambdas` after testing other ranges. The benefit of the structure of `even.more.lamdbas` is that exponentiation allowed me to explore a wide range of values for `lambda` with a higher level of granularity as lower values.

```
# regularization procedures: define overall average `mu` and bias penalty ranges to test
mu <- mean(edx.train$rating)
lambdas <- seq(0, 20, 0.1)
more.lambdas <- seq(0, 100, 5)
even.more.lambdas <- 10^seq(-2, 2, 0.1)
```

Regularization followed the same general procedure for each term. For the sake of brevity, I have summarized the procedure here:

1. Subtract the overall mean (`mu`) from each `rating` to calculate the difference.
2. Stratify by the predictor in question.
3. Determine the number of ratings in each stratum.
4. Calculate the sum of the difference between the mean (`mu`) and each `rating`.
5. Test a range of values for `lambda` (the regularization term).
6. Select the value for `lambda` that minimizes RMSE.
7. Store the regularized bias terms with the ideal value for `lambda` in a variable for future modeling.

Movie Bias Term Regularization

```
# find ideal lambda for movie regularization
movie.rmses <- sapply(even.more.lambdas, function(x) {
  edx.train %>%
    mutate(mu = mean(rating),
           y.mu.diff = rating - mu) %>%
    group_by(movieId) %>%
    summarise(n.movies = n(),
              sum.movie.diff = sum(y.mu.diff),
              b_lambda = sum.movie.diff / (n.movies + x)) %>%
    left_join(edx.test, by = "movieId") %>%
    mutate(pred = mu + b_lambda) %>%
    filter(!is.na(rating)) %>%
    summarise(rmse = RMSE(pred, rating)) %>%
    pull(rmse)
})

# cbind(lambdas, movie.rmses)
# cbind(more.lambdas, movie.rmses)
cbind(even.more.lambdas, movie.rmses) %>% kable()
```

even.more.lambdas	movie.rmses
0.0100000	0.9437139
0.0125893	0.9437137
0.0158489	0.9437135
0.0199526	0.9437132
0.0251189	0.9437128
0.0316228	0.9437124
0.0398107	0.9437119
0.0501187	0.9437112
0.0630957	0.9437104
0.0794328	0.9437094
0.1000000	0.9437082
0.1258925	0.9437068
0.1584893	0.9437051
0.1995262	0.9437030
0.2511886	0.9437007
0.3162278	0.9436980
0.3981072	0.9436949
0.5011872	0.9436917
0.6309573	0.9436882
0.7943282	0.9436847
1.0000000	0.9436815
1.2589254	0.9436790
1.5848932	0.9436776
1.9952623	0.9436783
2.5118864	0.9436821
3.1622777	0.9436904
3.9810717	0.9437054
5.0118723	0.9437297

even.more.lambdas	movie.rmses
6.3095734	0.9437664
7.9432823	0.9438199
10.0000000	0.9438952
12.5892541	0.9439986
15.8489319	0.9441375
19.9526231	0.9443209
25.1188643	0.9445590
31.6227766	0.9448642
39.8107171	0.9452505
50.1187234	0.9457342
63.0957344	0.9463336
79.4328235	0.9470691
100.0000000	0.9479632

```
l.movie <- even.more.lambdas[which.min(movie.rmses)]

# movie regularization
edx.train %>%
  mutate(mu = mean(rating),
         y.mu.diff = rating - mu) %>%
  group_by(movieId) %>%
  summarise(n.movies = n(),
            sum.movie.diff = sum(y.mu.diff),
            b_lambda = sum.movie.diff / (n.movies + l.movie)) %>%
  ungroup() %>%
  left_join(edx.test, by = "movieId") %>%
  mutate(pred = mu + b_lambda) %>%
  filter(!is.na(rating)) %>%
  summarise(rmse = RMSE(pred, rating)) %>%
  kable()
```

rmse
0.9436776

```
b.movie <- edx.train %>%
  mutate(mu = mean(rating),
         y.mu.diff = rating - mu) %>%
  group_by(movieId) %>%
  summarise(n.movies = n(),
            sum.movie.diff = sum(y.mu.diff),
            b_movie = sum.movie.diff / (n.movies + l.movie)) %>%
  ungroup()
```

The ideal `lambda` for regularization of the `movieId` bias term is 1.5848932.

User Bias Term Regularization

```r
# find ideal lambda for user regularization
user.rmses <- sapply(even.more.lambdas, function(x) {
  edx.train %>%
    left_join(b.movie, by = "movieId") %>%
    mutate(mu = mean(rating),
           y.mu.diff = rating - mu - b_movie) %>%
    group_by(userId) %>%
    summarise(n.users = n(),
              sum.user.diff = sum(y.mu.diff),
              b_lambda = sum.user.diff / (n.users + x)) %>%
    left_join(edx.test, by = "userId") %>%
    mutate(pred = mu + b_lambda) %>%
    filter(!is.na(rating)) %>%
    summarise(rmse = RMSE(pred, rating)) %>%
    pull(rmse)
})

cbind(even.more.lambdas, user.rmses) %>% kable()
```

even.more.lambdas	user.rmses
0.0100000	0.9951257
0.0125893	0.9951249
0.0158489	0.9951239
0.0199526	0.9951226
0.0251189	0.9951211
0.0316228	0.9951191
0.0398107	0.9951166
0.0501187	0.9951135
0.0630957	0.9951096
0.0794328	0.9951047
0.1000000	0.9950985
0.1258925	0.9950909
0.1584893	0.9950813
0.1995262	0.9950693
0.2511886	0.9950544
0.3162278	0.9950360
0.3981072	0.9950133
0.5011872	0.9949853
0.6309573	0.9949513
0.7943282	0.9949101
1.0000000	0.9948608
1.2589254	0.9948026
1.5848932	0.9947350
1.9952623	0.9946584
2.5118864	0.9945742
3.1622777	0.9944854
3.9810717	0.9943975
5.0118723	0.9943188
6.3095734	0.9942616
7.9432823	0.9942420
10.0000000	0.9942804
12.5892541	0.9944013

even.more.lambdas	user.rmses
15.8489319	0.9946321
19.9526231	0.9950026
25.1188643	0.9955431
31.6227766	0.9962832
39.8107171	0.9972506
50.1187234	0.9984692
63.0957344	0.9999582
79.4328235	1.0017312
100.0000000	1.0037942

```
l.user <- even.more.lambdas[which.min(user.rmses)]

# user regularization
edx.train %>%
  left_join(b.movie, by = "movieId") %>%
  mutate(mu = mean(rating),
         y.mu.diff = rating - mu - b_movie) %>%
  group_by(userId) %>%
  summarise(n.users = n(),
            sum.user.diff = sum(y.mu.diff),
            b_lambda = sum.user.diff / (n.users + l.user)) %>%
  left_join(edx.test, by = "userId") %>%
  mutate(pred = mu + b_lambda) %>%
  filter(!is.na(rating)) %>%
  summarise(rmse = RMSE(pred, rating)) %>%
  kable()
```

rmse
0.994242

```
b.user <- edx.train %>%
  left_join(b.movie, by = "movieId") %>%
  mutate(mu = mean(rating),
         y.mu.diff = rating - mu - b_movie) %>%
  group_by(userId) %>%
  summarise(n.users = n(),
            sum.user.diff = sum(y.mu.diff),
            b_user = sum.user.diff / (n.users + l.user)) %>%
  ungroup()
```

The ideal `lambda` for regularization of the `userId` bias term is 7.9432823.

Genre Bias Term Regularization

```
# find ideal lambda for genre regularization
genre.rmses <- sapply(even.more.lambdas, function(x) {
  edx.train %>%
```

```
    left_join(b.movie, by = "movieId") %>%
    left_join(b.user, by = "userId") %>%
    mutate(mu = mean(rating),
           y.mu.diff = rating - mu - b_movie - b_user) %>%
    group_by(genres) %>%
    summarise(n.genres = n(),
              sum.genre.diff = sum(y.mu.diff),
              b_lambda = sum.genre.diff / (n.genres + x)) %>%
    left_join(edx.test, by = "genres") %>%
    mutate(pred = mu + b_lambda) %>%
    filter(!is.na(rating)) %>%
    summarise(rmse = RMSE(pred, rating)) %>%
    pull(rmse)
})

cbind(even.more.lambdas, genre.rmses) %>% kable()
```

even.more.lambdas	genre.rmses
0.0100000	1.059639
0.0125893	1.059639
0.0158489	1.059639
0.0199526	1.059639
0.0251189	1.059639
0.0316228	1.059639
0.0398107	1.059639
0.0501187	1.059639
0.0630957	1.059639
0.0794328	1.059639
0.1000000	1.059639
0.1258925	1.059639
0.1584893	1.059639
0.1995262	1.059639
0.2511886	1.059639
0.3162278	1.059639
0.3981072	1.059639
0.5011872	1.059639
0.6309573	1.059639
0.7943282	1.059639
1.0000000	1.059639
1.2589254	1.059639
1.5848932	1.059639
1.9952623	1.059639
2.5118864	1.059639
3.1622777	1.059639
3.9810717	1.059639
5.0118723	1.059639
6.3095734	1.059639
7.9432823	1.059639
10.0000000	1.059639
12.5892541	1.059639
15.8489319	1.059639
19.9526231	1.059639

even.more.lambdas	genre.rmses
25.1188643	1.059639
31.6227766	1.059639
39.8107171	1.059639
50.1187234	1.059639
63.0957344	1.059639
79.4328235	1.059639
100.0000000	1.059639

```
l.genre <- even.more.lambdas[which.min(genre.rmses)]

# genre regularization
edx.train %>%
  left_join(b.movie, by = "movieId") %>%
  left_join(b.user, by = "userId") %>%
  mutate(mu = mean(rating),
         y.mu.diff = rating - mu - b_movie - b_user) %>%
  group_by(genres) %>%
  summarise(n.genres = n(),
            sum.genre.diff = sum(y.mu.diff),
            b_lambda = sum.genre.diff / (n.genres + l.genre)) %>%
  left_join(edx.test, by = "genres") %>%
  mutate(pred = mu + b_lambda) %>%
  filter(!is.na(rating)) %>%
  summarise(rmse = RMSE(pred, rating)) %>%
  kable()
```

rmse
1.059639

```
b.genre <- edx.train %>%
  left_join(b.movie, by = "movieId") %>%
  left_join(b.user, by = "userId") %>%
  mutate(mu = mean(rating),
         y.mu.diff = rating - mu - b_movie - b_user) %>%
  group_by(genres) %>%
  summarise(n.genres = n(),
            sum.genre.diff = sum(y.mu.diff),
            b_genre = sum.genre.diff / (n.genres + l.genre)) %>%
  ungroup()
```

The ideal `lambda` for regularization of the `genres` bias term is 0.7943282.

Year Bias Term Regularization

```
# find ideal lambda for year regularization
year.rmses <- sapply(even.more.lambdas, function(x) {
  edx.train %>%
```

```
    left_join(b.movie, by = "movieId") %>%
    left_join(b.user, by = "userId") %>%
    left_join(b.genre, by = "genres") %>%
    mutate(mu = mean(rating),
           y.mu.diff = rating - mu - b_movie - b_user - b_genre) %>%
    group_by(year) %>%
    summarise(n.year = n(),
              sum.year.diff = sum(y.mu.diff),
              b_lambda = sum.year.diff / (n.year + x)) %>%
    left_join(edx.test, by = "year") %>%
    mutate(pred = mu + b_lambda) %>%
    filter(!is.na(rating)) %>%
    summarise(rmse = RMSE(pred, rating)) %>%
    pull(rmse)
})

cbind(even.more.lambdas, year.rmses) %>% kable()
```

even.more.lambdas	year.rmses
0.0100000	1.059873
0.0125893	1.059873
0.0158489	1.059873
0.0199526	1.059873
0.0251189	1.059873
0.0316228	1.059873
0.0398107	1.059873
0.0501187	1.059873
0.0630957	1.059873
0.0794328	1.059873
0.1000000	1.059873
0.1258925	1.059873
0.1584893	1.059873
0.1995262	1.059873
0.2511886	1.059873
0.3162278	1.059873
0.3981072	1.059873
0.5011872	1.059873
0.6309573	1.059873
0.7943282	1.059873
1.0000000	1.059873
1.2589254	1.059873
1.5848932	1.059873
1.9952623	1.059873
2.5118864	1.059873
3.1622777	1.059873
3.9810717	1.059873
5.0118723	1.059873
6.3095734	1.059873
7.9432823	1.059873
10.0000000	1.059873
12.5892541	1.059873
15.8489319	1.059873

even.more.lambdas	year.rmses
19.9526231	1.059874
25.1188643	1.059874
31.6227766	1.059874
39.8107171	1.059874
50.1187234	1.059875
63.0957344	1.059875
79.4328235	1.059876
100.0000000	1.059877

```r
l.year <- even.more.lambdas[which.min(year.rmses)]

# year regularization
edx.train %>%
  left_join(b.movie, by = "movieId") %>%
  left_join(b.user, by = "userId") %>%
  left_join(b.genre, by = "genres") %>%
  mutate(mu = mean(rating),
         y.mu.diff = rating - mu - b_movie - b_user - b_genre) %>%
  group_by(year) %>%
  summarise(n.year = n(),
            sum.year.diff = sum(y.mu.diff),
            b_year = sum.year.diff / (n.year + l.year)) %>%
  left_join(edx.test, by = "year") %>%
  mutate(pred = mu + b_year) %>%
  filter(!is.na(rating)) %>%
  summarise(rmse = RMSE(pred, rating)) %>%
  kable()
```

rmse
1.059873

```r
b.year <-edx.train %>%
  left_join(b.movie, by = "movieId") %>%
  left_join(b.user, by = "userId") %>%
  left_join(b.genre, by = "genres") %>%
  mutate(mu = mean(rating),
         y.mu.diff = rating - mu - b_movie - b_user - b_genre) %>%
  group_by(year) %>%
  summarise(n.year = n(),
            sum.year.diff = sum(y.mu.diff),
            b_year = sum.year.diff / (n.year + l.year)) %>%
  ungroup()
```

The ideal `lambda` for regularization of the `year` bias term is 0.01.

Results

Model Evaluation

Naive Model ("Guessing")

I started by modeling random guesses for the `rating` variable by sampling from a discrete uniform distribution between 0.5 and 5 inclusive with replacement. Random guessing yields a RMSE of greater than `1.9`. This will serve as our baseline for evaluation.

```
# naive model ("guessing")
guesses <- sample(x = seq(0.5, 5, 0.5),
                  size = nrow(edx.test),
                  replace = T)
RMSE(guesses, edx.test$rating)
```

```
## [1] 1.941357
```

Movie Bias Model

After adding in a regularized bias term for `movieId`, we get a significantly improves RMSE of `0.9436776`.

```
edx.test %>%
  left_join(b.movie, by = "movieId") %>%
  mutate(pred = mu + b_movie) %>%
  filter(!is.na(pred)) %>%
  summarise(rmse = RMSE(rating, pred)) %>%
  kable()
```

rmse
0.9436776

Movie + User Bias Model

Including a regularized bias term for `userId` further improves the RMSE of our predictions to `0.865678`.

```
edx.test %>%
  left_join(b.movie, by = "movieId") %>%
  left_join(b.user, by = "userId") %>%
  mutate(pred = mu + b_movie + b_user) %>%
  filter(!is.na(pred)) %>%
  summarise(rmse = RMSE(rating, pred)) %>%
  kable()
```

rmse
0.865678

Movie + User + Genre Bias Model

Adding the regularized bias term for `genres` provides a very small improvement to RMSE: `0.8655764`.

```
edx.test %>%
  left_join(b.genre, by = "genres") %>%
  left_join(b.movie, by = "movieId") %>%
  left_join(b.user, by = "userId") %>%
  mutate(pred = mu + b_movie + b_user + b_genre) %>%
  filter(!is.na(pred)) %>%
  summarise(rmse = RMSE(rating, pred)) %>%
  kable()
```

rmse
0.8655764

Movie + User + Genre + Year Bias Model

Adding the final regularized bias term for `year` improves our RMSE by another small margin: `0.8653371`

```
edx.test %>%
  left_join(b.year, by = "year") %>%
  left_join(b.genre, by = "genres") %>%
  left_join(b.movie, by = "movieId") %>%
  left_join(b.user, by = "userId") %>%
  mutate(pred = mu + b_movie + b_user + b_genre + b_year) %>%
  filter(!is.na(pred)) %>%
  summarise(rmse = RMSE(rating, pred)) %>%
  kable()
```

rmse
0.8653371

Our final model is as follows: $rating = \mu + b_{movie}(\lambda) + b_{user}(\lambda) + b_{genre}(\lambda) + b_{year}(\lambda)$ where `mu` is the overall average of `rating` in the training set, and `b_predictor(lambda)` are the regularized bias terms for each of the selected predictors.

Final Model Performance

When tested against the pre-processed `final_holdout_test` set, the model yields a final RMSE of `0.8653659` (less than 0.86549).

```
# final testing
final_holdout_test %>%
  mutate(year = factor(str_match(final_holdout_test$title, "\\((\\d{4})\\)$")[,2]),
         genres = factor(str_match(final_holdout_test$genres, "^[^|]+")),
         userId = factor(userId),
         movieId = factor(movieId)) %>%
```

```
left_join(b.year, by = "year") %>%
left_join(b.genre, by = "genres") %>%
left_join(b.movie, by = "movieId") %>%
left_join(b.user, by = "userId") %>%
mutate(pred = mu + b_movie + b_user + b_genre + b_year) %>%
filter(!is.na(pred)) %>%
summarise(final.rmse = RMSE(rating, pred)) %>%
kable()
```

final.rmse
0.8653659

Conclusion

Summary

In conclusion, a predictive model for `ratings` that includes regularized bias terms for the `movieId`, `userId`, `genres`, and `year` predictors yielded the lowest RMSE and was therefore the best model. The model can be represented with the following equation: $rating = \mu + b_{movie}(\lambda) + b_{user}(\lambda) + b_{genre}(\lambda) + b_{year}(\lambda)$

Limitations and Future Work

Due to memory (RAM) limitations, some of the `caret` library's more powerful models were not available to me as options. With greater processing power, I would like to compare the results of the model I have developed to those of a k-nearest neighbors classification model and a random forest model. Moreover, I would like to implement k-folds cross-validation to identify the ideal tuning parameters for each of the models (e.g. `k` and `mtry` respectively).