# Project MovieLens

Paul Uhn

2023-11-11

## Introduction

This document describes my project for building the algorithm for a movie recommendation system. A recommendation system can be useful because it can take a user's ratings for some items and make recommendations by predicting other items the user would likely give high ratings for.

Netflix uses a recommendation system to predict if a user will like a specific movie or tv show. Previously, the ratings were based on the number of stars (1-5) but currently it uses a simpler system of thumbs up, two thumbs up and thumbs down for user ratings.

In 2006, Netflix held an open competition for improving their recommendation algorithm and awarded the grand prize in September 2009. Due to privacy concerns, those datasets are no longer available. So for this project, I used the MovieLens 10M dataset provided by the GroupLens research lab. This dataset provides 10 million ratings - 10,000 movies rated by 72,000 users.

The final algorithm was developed in several steps starting with the most trivial one: the mean over all user-movie ratings. I selected and analyzed additional effects before including them as part of the algorithm's predictors. As part of this analysis, I kept a running table of the root mean squared error (RMSE) values. The best way to think of RMSE values is golf scores - the lower the better.

- Netflix Prize: https://en.wikipedia.org/wiki/Netflix_Prize
- Summary of the winning algorithm: http://blog.echen.me/2011/10/24/winning-the-netflix-prize-a-summary/
- Explanation of the winning algorithm: https://www2.seas.gwu.edu/~simhaweb/champalg/cf/papers/KorenBellKor2009.pdf
- MovieLens datasets: https://grouplens.org/datasets/movielens/

## Data Wrangling

First, the datasets need to be downloaded and converted into a usable format for analysis. This process is called `data wrangling`.

```
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),
         date = as_datetime(timestamp)) |>
  select(userId, movieId, rating, date)

movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies |>
  separate_wider_regex(title, c(title = ".*", " \\(", year = "\\d{4}", "\\)")) |>
  mutate(movieId = as.integer(movieId),
         year = as.integer(year))

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

Second, the datasets need to be split into training and test sets. The final hold-out test set will be 10% of the data and will **ONLY** be used to evaluate the RMSE of the final algorithm.

```
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
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,
                     by = join_by(userId, movieId, rating, date, title, genres))
edx <- rbind(edx, removed)
```

Here's the final training set `edx` with 9 million rows:

```
edx |> as_tibble()
```

```
## # A tibble: 9,000,055 x 7
##    userId movieId rating date                title                     year genres
##     <int>   <int>  <dbl> <dttm>              <chr>                    <int> <chr>
##  1      1     122      5 1996-08-02 11:24:06 Boomerang                 1992 Comed~
##  2      1     185      5 1996-08-02 10:58:45 Net, The                  1995 Actio~
##  3      1     292      5 1996-08-02 10:57:01 Outbreak                  1995 Actio~
##  4      1     316      5 1996-08-02 10:56:32 Stargate                  1994 Actio~
##  5      1     329      5 1996-08-02 10:56:32 Star Trek: Generations    1994 Actio~
##  6      1     355      5 1996-08-02 11:14:34 Flintstones, The          1994 Child~
##  7      1     356      5 1996-08-02 11:00:53 Forrest Gump              1994 Comed~
##  8      1     362      5 1996-08-02 11:21:25 Jungle Book, The          1994 Adven~
##  9      1     364      5 1996-08-02 11:01:47 Lion King, The            1994 Adven~
## 10      1     370      5 1996-08-02 11:16:36 Naked Gun 33 1/3: The~    1994 Actio~
## # i 9,000,045 more rows
```

We can also see the number of unique users and unique movies:

```
edx |> summarize(n_users = n_distinct(userId),
                 n_movies = n_distinct(movieId))
```

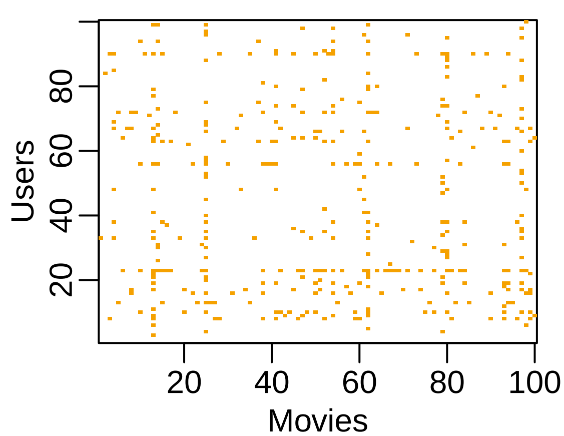
```
##   n_users n_movies
## 1   69878    10677
```

But multiplying these two numbers results in almost 750 million ratings! Given that we actually have only 9 million ratings, we can guess that not every user rated every movie. Here's a subset that proves our hunch:

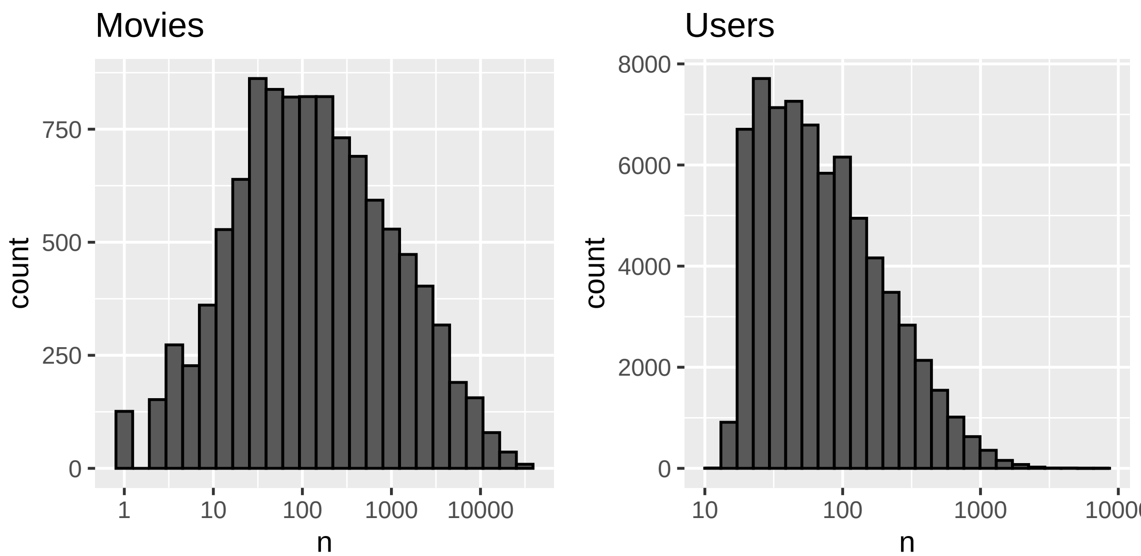
| userId | Pulp Fiction | Jurassic Park | Silence of the Lambs | Forrest Gump |
|---:|---:|---:|---:|---:|
| 18 | 5 | 3 | 5 | NA |
| 19 | NA | 1 | NA | 4 |
| 22 | 5 | 4 | 5 | NA |
| 23 | 4 | NA | 5 | NA |
| 30 | NA | 4 | 5 | 5 |
| 34 | 1 | 4 | 5 | 1 |

Note the `NA` as missing user ratings. As well as the different movie ratings per user.

Just to see how *sparse* the data really is, here's a random sample of 100 movies and 100 users with the color indicating a user rating.



Here's another way to look at the data: some movies are blockbusters and watched by millions. And some users are just way more active than others.




## Analysis

To aid in our analysis, I split the `edx` into training and test sets by assigning 20% of the ratings by each user to the test set:

```
set.seed(2023)
indexes <- split(1:nrow(edx), edx$userId)
test_ind <- sapply(indexes, function(ind) sample(ind, ceiling(length(ind)*.2))) |>
  unlist(use.names = TRUE) |> sort()
test_set <- edx[test_ind,]
train_set <- edx[-test_ind,]
```

And we remove any movies that are **not** in both training and test sets:

```
test_set <- test_set |>
  semi_join(train_set, by = "movieId")
train_set <- train_set |>
  semi_join(test_set, by = "movieId")
```

Finally, we create a matrix of users (row) and movies (column):

```
y <- train_set |>
  select(movieId, userId, rating) |>
  pivot_wider(names_from = movieId, values_from = rating)
rnames <- y$userId
y <- as.matrix(y[,-1])
rownames(y) <- rnames
```

As well as a table to map the movie ids to titles:

```
movie_map <- train_set |>
  select(movieId, title) |>
  distinct(movieId, .keep_all = TRUE)
```

### RMSE

To evaluate how well the algorithm is performing, we will be using the RMSE on the test set. The RMSE is as defined:

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

where $y_{u,i}$ is the rating for movie $i$ by user $u$, our prediction for the rating is $\hat{y}_{u,i}$ and $N$ is the number of user/movie combinations.

Another way to think of it is to get all the differences between our prediction and the actual user ratings and square them. Then get the square root of the mean. In R, we define the RMSE as a function:

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

### First Algorithm

The first algorithm should predict a constant value for the rating regardless of movie or user:

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

where $\varepsilon_{i,u}$ is the independent errors sampled from the same distribution centered at zero and $\mu$ is the *true* rating for all movies.

In this case, $\mu$ becomes the average rating of all movies across all users.

```
mu <- mean(y, na.rm = TRUE)
mu
```

```
## [1] 3.512135
```

Using $\mu$ to predict all unknown ratings gives the following RMSE:

```
trivial_rmse <- RMSE(test_set$rating, mu)
trivial_rmse
```

```
## [1] 1.060691
```

Here's our results so far:

| method | RMSE |
|---|---|
| Just the average | 1.060691 |

### Movie Effects

We know from past experience that some movies are rated higher than others. This bias, $b$, can therefore be added to our algorithm:

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

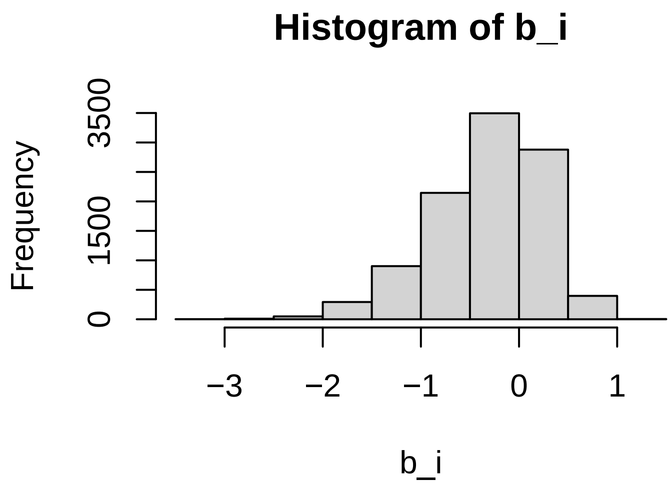
where $b_i$ is the movie bias effect for movie $i$.

We can then compute that $b_i$ is just the average of $y_{u,i} - \mu$ for each movie $i$:

```
b_i <- colMeans(y - mu, na.rm = TRUE)
```

And we can see that these estimates vary substantially:

```
hist(b_i)
```




Note that $\mu = 3.5$ so $b_i = 1.5$ implies a perfect five star rating.

Let's see how our predictions improve using $b_i$:

```
fit_movies <- data.frame(movieId = as.integer(colnames(y)),
                         mu = mu,
                         b_i = b_i)
movie_effect_rmse <- left_join(test_set, fit_movies, by = "movieId") |>
  mutate(pred = mu + b_i) |>
  summarize(rmse = RMSE(rating, pred)) |>
  pull(rmse)
movie_effect_rmse
```

```
## [1] 0.9445651
```

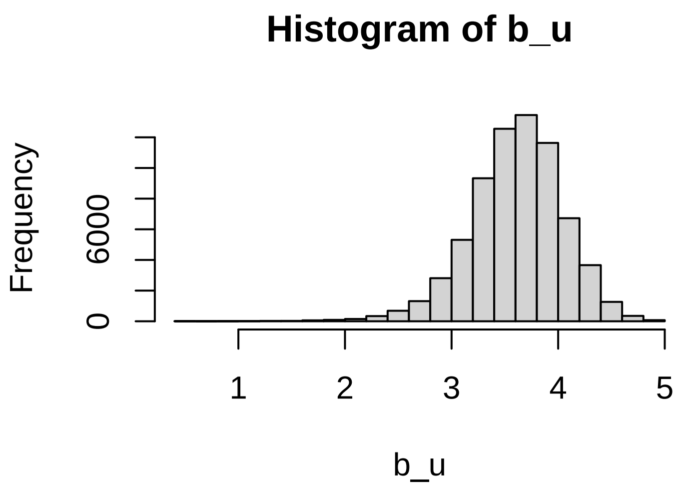
And our results so far:

| method | RMSE |
|---|---|
| Just the average | 1.0606908 |
| Movie effect | 0.9445651 |

**User Effects**

Let's compute the average rating for user $u$ to check for any user bias:

```
b_u <- rowMeans(y, na.rm = TRUE)
hist(b_u)
```




Note the substantial variability across users - there is definitely user bias here:

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

where $b_u$ is the user effect bias for user $u$.

We can compute $b_u$ as the average of $y_{u,i} - \mu - b_i$ for each user $u$:

```
b_u <- rowMeans(sweep(y - mu, 2, b_i), na.rm = TRUE)
```

We can then make our predictions:

```
fit_users <- data.frame(userId = as.integer(rownames(y)),
                        b_u = b_u)
user_effect_rmse <- left_join(test_set, fit_movies, by = "movieId") |>
  left_join(fit_users, by = "userId") |>
  mutate(pred = mu + b_i + b_u) |>
  summarize(rmse = RMSE(rating, pred)) |>
  pull(rmse)
user_effect_rmse
```

```
## [1] 0.8669772
```

And here's where we are so far:

| method | RMSE |
|---|---|
| Just the average | 1.0606908 |
| Movie effect | 0.9445651 |
| Movie + User effect | 0.8669772 |

### Regularization

Based on our estimates of the movie effect $b_i$, let's take a look at the top movies scoring above $4\frac{1}{2}$ stars:

```
n <- colSums(!is.na(y))
fit_movies$n <- n
best <- fit_movies |> left_join(movie_map, by = "movieId") |>
  mutate(average_rating = mu + b_i) |>
  filter(average_rating > 4.5 & n > 1)
test_set |>
  group_by(movieId) |>
  summarize(test_set_averge_rating = mean(rating)) |>
  right_join(best, by = "movieId") |>
  select(title, average_rating, n, test_set_averge_rating)
```

```
## # A tibble: 4 x 4
##   title                                average_rating     n test_set_averge_rating
##   <chr>                                         <dbl> <dbl>                  <dbl>
## 1 Who's Singin' Over There? (a.k.a.~             5        2                    4.5
## 2 Life of Oharu, The (Saikaku ichid~             4.75     2                    4
## 3 Human Condition II, The (Ningen n~             4.83     3                    4.5
## 4 Carmen                                         4.67     3                    2.5
```

These all seem like obscure movies. The reason being they were highly rated by very few users. To fix this, we need to penalize large estimates formed by small sample sizes and reduce the penalty as the sample size grows.

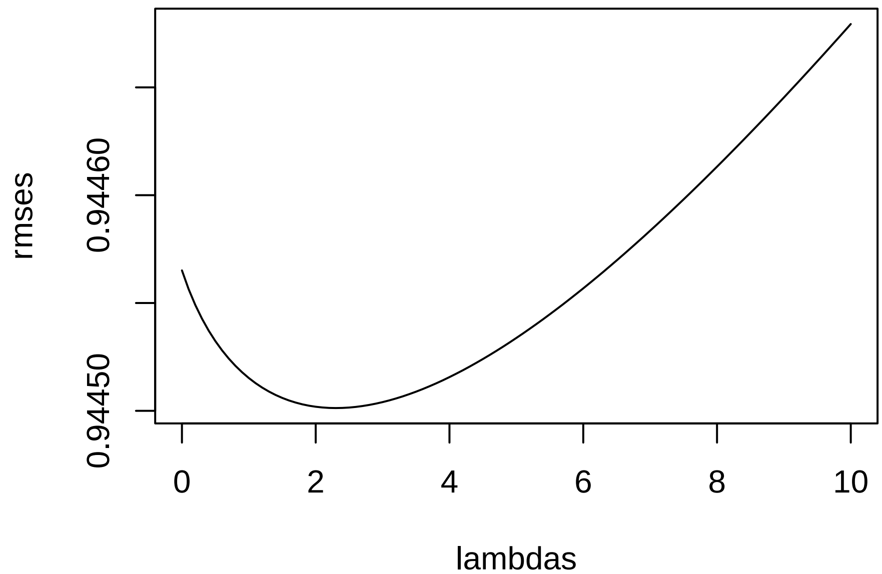
$$b_i(\lambda) = \frac{1}{\lambda + n_i} \sum_{u=1}^{n_i} (y_{u,i} - \mu)$$

where $\lambda$ is the penalty, and $n_i$ is the number of ratings made for movie $i$. So when sample size $n_i$ is very large, the penalty $\lambda$ is effectively ignored since $n_i + \lambda \approx n_i$. But when $n_i$ is small, it causes $b_i(\lambda)$ to shrink to 0.

```
lambdas <- seq(0, 10, 0.1)
sums <- colSums(y - mu, na.rm = TRUE)
rmses <- sapply(lambdas, function(lambda) {
  b_i <- sums / (n + lambda)
  fit_movies$b_i <- b_i
  left_join(test_set, fit_movies, by = "movieId") |>
    mutate(pred = mu + b_i) |>
    summarize(rmse = RMSE(rating, pred)) |>
    pull(rmse)
})
```

We then select the value that minimizes the RMSE:

```
plot(lambdas, rmses, type = "l")
```




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

```
## [1] 2.3
```

After selecting a $\lambda$, we can compute the regularized estimates:

```
fit_movies$b_i_reg <- sums / (n + lambda)
```

Now, let's look at the top 5 best movies based on the penalized $b_i(\lambda)$:

```
## # A tibble: 5 x 4
##   title                     average_rating     n test_set_averge_rating
##   <chr>                              <dbl> <dbl>                  <dbl>
## 1 Usual Suspects, The                 4.36 17191                   4.39
## 2 Shawshank Redemption, The           4.46 22241                   4.45
## 3 Schindler's List                    4.37 18443                   4.35
## 4 Godfather, The                      4.42 14168                   4.40
## 5 Rear Window                         4.32  6286                   4.31
```

Yes! Much better! Did we improve our results?

```
reg_movie_rmse <- left_join(test_set, fit_movies, by = "movieId") |>
  mutate(pred = mu + b_i_reg) |>
  summarize(rmse = RMSE(rating, pred)) |> pull(rmse)
reg_movie_rmse
```

```
## [1] 0.9445013
```

```
reg_movie_user_rmse <- left_join(test_set, fit_movies, by = "movieId") |>
  left_join(fit_users, by = "userId") |>
  mutate(pred = mu + b_i_reg + b_u) |>
  summarize(rmse = RMSE(rating, pred)) |> pull(rmse)
reg_movie_user_rmse
```

```
## [1] 0.8668415
```

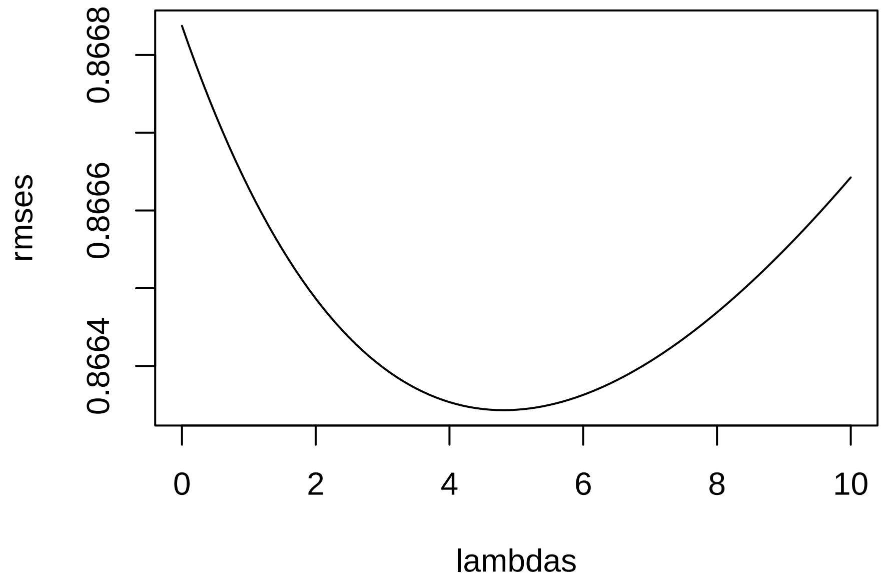
The penalized estimates did improve our RMSE values:

| method | RMSE |
|---|---:|
| Just the average | 1.0606908 |
| Movie effect | 0.9445651 |
| Regularized Movie effect | 0.9445013 |
| Movie + User effect | 0.8669772 |
| Regularized Movie + User effect | 0.8668415 |

Given the improvements by regularizing the movie effect, let's look at regularizing the user effect next.

```
m <- rowSums(!is.na(y))
fit_users$m <- m
lambdas <- seq(0, 10, 0.1)
sums <- rowSums(sweep(y - mu, 2, fit_movies$b_i_reg), na.rm = TRUE)
rmses <- sapply(lambdas, function(lambda) {
  b_u <- sums / (m + lambda)
  fit_users$b_u <- b_u
  left_join(test_set, fit_movies, by = "movieId") |>
    left_join(fit_users, by = "userId") |>
    mutate(pred = mu + b_i_reg + b_u) |>
    summarize(rmse = RMSE(rating, pred)) |>
    pull(rmse)
})
```

```r
plot(lambdas, rmses, type = "l")
```




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

```
## [1] 4.8
```

Now, let's regularize the user effect with the selected $\lambda$:

```r
fit_users$b_u_reg <- sums / (m + lambda)
reg_user_rmse <- left_join(test_set, fit_movies, by = "movieId") |>
  left_join(fit_users, by = "userId") |>
  mutate(pred = mu + b_i_reg + b_u_reg) |>
  summarize(rmse = RMSE(rating, pred)) |>
  pull(rmse)
reg_user_rmse
```

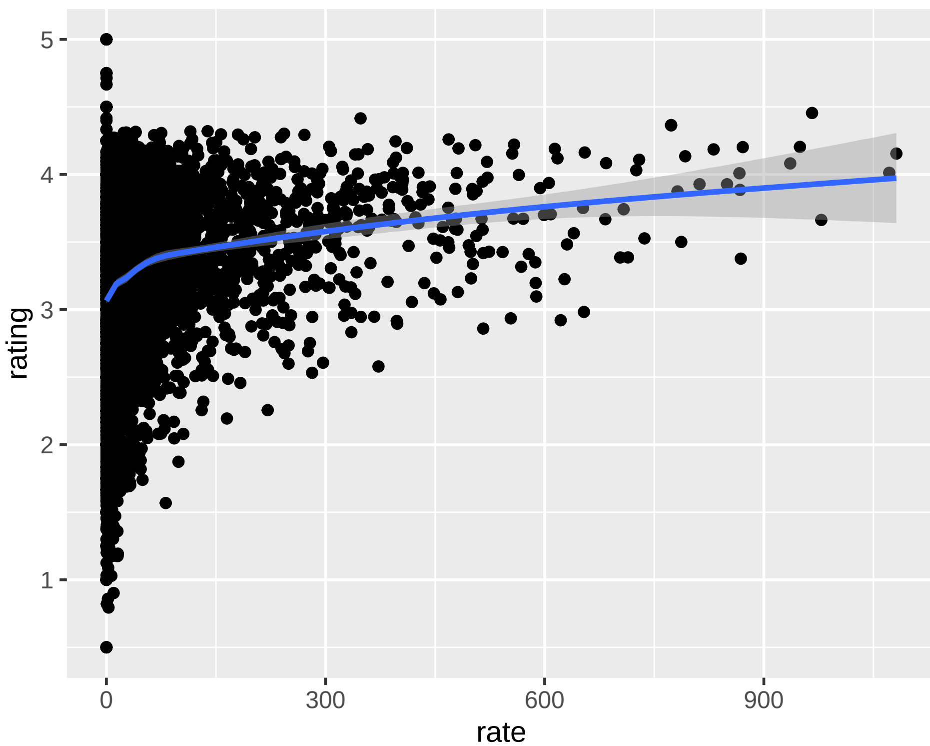
```
## [1] 0.8663432
```

| method | RMSE |
|---|---|
| Just the average | 1.0606908 |
| Movie effect | 0.9445651 |
| Regularized Movie effect | 0.9445013 |
| Movie + User effect | 0.8669772 |
| Regularized Movie + User effect | 0.8668415 |
| Reg Movie + Reg User effect | 0.8663432 |

**Rate Effects**

Is there any relationship between how often a movie is rated vs its ratings? Our intuition tells us that, on average, more people tend to rate good movies vs bad movies. But can we confirm this by exploring the `edx` dataset?

```
edx |>
  group_by(movieId) |>
  summarize(n = n(),
            years = 2023 - first(year),
            title = title[1],
            rating = mean(rating)) |>
  mutate(rate = n / years) |>
  ggplot(aes(rate, rating)) +
  geom_point() +
  geom_smooth()
```




So, we can see the trend confirms that the more frequently a movie is rated, the more likely it will have above average ratings.

First, let's start with just the rate bias effect, $b_r$:

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

We prep the training set:

```
rate_to_rating <- train_set |>
  mutate(rating = rating - mu) |>
  group_by(movieId) |>
  summarize(n = n(),
            years = 2023 - first(year),
            title = title[1],
            rating = mean(rating)) |>
  mutate(rate = n / years)
```

And to aid in our computations, we set aside a mapping between movie id and its rate:

```
rate_map <- rate_to_rating |>
  select(movieId, rate)
```

Next, we prep our test set:

```
test_set_rate <- test_set |>
  left_join(rate_map, by = "movieId")
```

Here we want to compute $b_r$ and compare three different methods:

```
fit_rate_glm <- train(rating ~ rate, method = "glm", data = rate_to_rating)
fit_rate_gam <- train(rating ~ rate, method = "gamLoess", data = rate_to_rating)
fit_rate_knn <- train(rating ~ rate, method = "knn", data = rate_to_rating)

b_r_glm <- predict(fit_rate_glm, test_set_rate)
b_r_gam <- predict(fit_rate_gam, test_set_rate)
b_r_knn <- predict(fit_rate_knn, test_set_rate)

data.frame(rating = test_set_rate$rating,
           pred_glm = mu + b_r_glm,
           pred_gam = mu + b_r_gam,
           pred_knn = mu + b_r_knn) |>
  summarize(rmse_glm = RMSE(rating, pred_glm),
            rmse_gam = RMSE(rating, pred_gam),
            rmse_knn = RMSE(rating, pred_knn))
```

```
##   rmse_glm rmse_gam rmse_knn
## 1 1.061823 1.048869 1.037032
```

Out of the three, only two performed better than the trivial algorithm but all performed worse than the movie effect. So instead, let's add $b_r$ to our best performing algorithm so far:

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

```r
# prep traing set
rate_to_rating <- train_set |>
  left_join(fit_movies, by = "movieId") |>
  left_join(fit_users, by = "userId") |>
  mutate(rating = rating - mu - b_i_reg - b_u_reg) |>
  group_by(movieId) |>
  summarize(n = n(),
            years = 2023 - first(year),
            title = title[1],
            rating = mean(rating)) |>
  mutate(rate = n / years)

# prep test set
test_set_rate <- test_set |>
  left_join(fit_movies, by = "movieId") |>
  left_join(fit_users, by = "userId") |>
  left_join(rate_map, by = "movieId")

# compare training methods
fit_rate_glm <- train(rating ~ rate, method = "glm", data = rate_to_rating)
fit_rate_gam <- train(rating ~ rate, method = "gamLoess", data = rate_to_rating)
fit_rate_knn <- train(rating ~ rate, method = "knn", data = rate_to_rating)

b_r_glm <- predict(fit_rate_glm, test_set_rate)
b_r_gam <- predict(fit_rate_gam, test_set_rate)
b_r_knn <- predict(fit_rate_knn, test_set_rate)

test_set_b_r <- test_set_rate |>
  mutate(pred_glm = mu + b_i_reg + b_u_reg + b_r_glm,
         pred_gam = mu + b_i_reg + b_u_reg + b_r_gam,
         pred_knn = mu + b_i_reg + b_u_reg + b_r_knn) |>
  summarize(rmse_glm = RMSE(rating, pred_glm),
            rmse_gam = RMSE(rating, pred_gam),
            rmse_knn = RMSE(rating, pred_knn))
test_set_b_r
```

```
##    rmse_glm  rmse_gam  rmse_knn
## 1 0.8664498 0.8660906 0.8660144
```

```r
min(test_set_b_r) # knn was best
```

```
## [1] 0.8660144
```

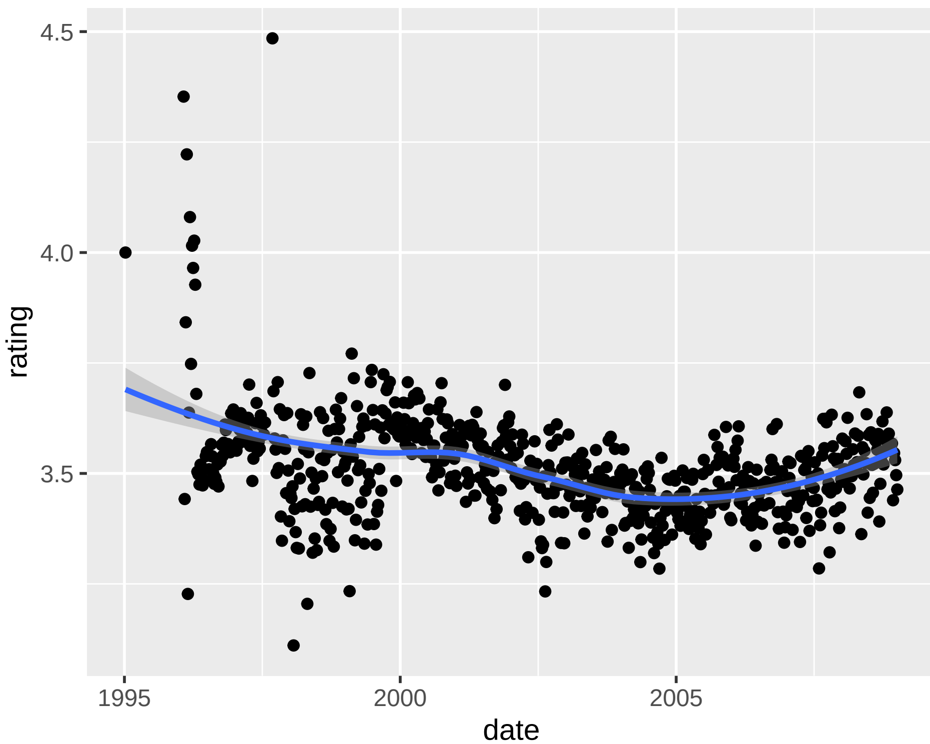
Let's add that to our RMSE results:

| method | RMSE |
|---|---|
| Just the average | 1.0606908 |
| Movie effect | 0.9445651 |
| Regularized Movie effect | 0.9445013 |
| Movie + User effect | 0.8669772 |
| Regularized Movie + User effect | 0.8668415 |
| Reg Movie + Reg User effect | 0.8663432 |
| Reg Movie + Reg User + Rate effect | 0.8660144 |

### Date Effects

What about *when* the user rated the movie? Could there be some relationship between the average rating per week vs date? Let's again use the `edx` dataset to find a trend, if any:

```
edx |>
  mutate(date = round_date(date, unit = "week")) |>
  group_by(date) |>
  summarize(rating = mean(rating)) |>
  ggplot(aes(date, rating)) +
  geom_point() +
  geom_smooth()
```




From the graph, we can see there appears to be *some* relationship that changes direction over time. One explanation for this could be the timing of award shows like the Oscars and the Golden Globes causing viewers to watch and rate that year's best picture. Another, older movies become popular again when their actors appear in new movies or sequels, etc.

Again, let's first start with just the date bias effect, $b_d$:

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

```r
# prep the training set
date_to_rating <- train_set |>
  mutate(rating = rating - mu,
         date = round_date(date, unit = "week")) |>
  group_by(date) |>
  summarize(rating = mean(rating))

# prep the test set
test_set_date <- test_set |>
  mutate(date = round_date(date, unit = "week"))
```

And again, we compare three methods for calculating $b_d$:

```r
fit_date_glm <- train(rating ~ date, method = "glm", data = date_to_rating)
fit_date_gam <- train(rating ~ date, method = "gamLoess", data = date_to_rating)
fit_date_knn <- train(rating ~ date, method = "knn", data = date_to_rating)

b_d_glm <- predict(fit_date_glm, test_set_date)
b_d_gam <- predict(fit_date_gam, test_set_date)
b_d_knn <- predict(fit_date_knn, test_set_date)

data.frame(rating = test_set_date$rating,
           pred_glm = mu + b_d_glm,
           pred_gam = mu + b_d_gam,
           pred_knn = mu + b_d_knn) |>
  summarize(rmse_glm = RMSE(rating, pred_glm),
            rmse_gam = RMSE(rating, pred_gam),
            rmse_knn = RMSE(rating, pred_knn))
```

```
##   rmse_glm rmse_gam rmse_knn
## 1 1.060066 1.059555 1.058381
```

This time, all three performed (slightly) better than the trivial algorithm but worse than the better algorithms. So, again, we add $b_d$ as a predictor to our best algorithm instead:

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

```r
# prep training set
date_to_rating <- train_set |>
  left_join(fit_movies, by = "movieId") |>
  left_join(fit_users, by = "userId") |>
  left_join(rate_map, by = "movieId")
date_to_rating <- date_to_rating |>
  mutate(b_r = predict(fit_rate, newdata = date_to_rating),
         rating = rating - mu - b_i_reg - b_u_reg - b_r,
         date = round_date(date, unit = "week")) |>
  group_by(date) |>
  summarize(rating = mean(rating))

# prep test set
test_set_date <- test_set |>
  left_join(fit_movies, by = "movieId") |>
  left_join(fit_users, by = "userId") |>
  left_join(rate_map, by = "movieId")
test_set_date <- test_set_date |>
```

```r
  mutate(b_r = predict(fit_rate, newdata = test_set_date),
         date = round_date(date, unit = "week"))

# compare training methods
fit_date_glm <- train(rating ~ date, method = "glm", data = date_to_rating)
fit_date_gam <- train(rating ~ date, method = "gamLoess", data = date_to_rating)
fit_date_knn <- train(rating ~ date, method = "knn", data = date_to_rating)

b_d_glm <- predict(fit_date_glm, test_set_date)
b_d_gam <- predict(fit_date_gam, test_set_date)
b_d_knn <- predict(fit_date_knn, test_set_date)

test_set_b_d <- test_set_date |>
  mutate(pred_glm = mu + b_i_reg + b_u_reg + b_r + b_d_glm,
         pred_gam = mu + b_i_reg + b_u_reg + b_r + b_d_gam,
         pred_knn = mu + b_i_reg + b_u_reg + b_r + b_d_knn) |>
  summarize(rmse_glm = RMSE(rating, pred_glm),
            rmse_gam = RMSE(rating, pred_gam),
            rmse_knn = RMSE(rating, pred_knn))
test_set_b_d
```

```
##    rmse_glm  rmse_gam  rmse_knn
## 1 0.8659896 0.8660021 0.8659442
```

```r
min(test_set_b_d) # knn was best
```

```
## [1] 0.8659442
```

Let's update our RMSE table:

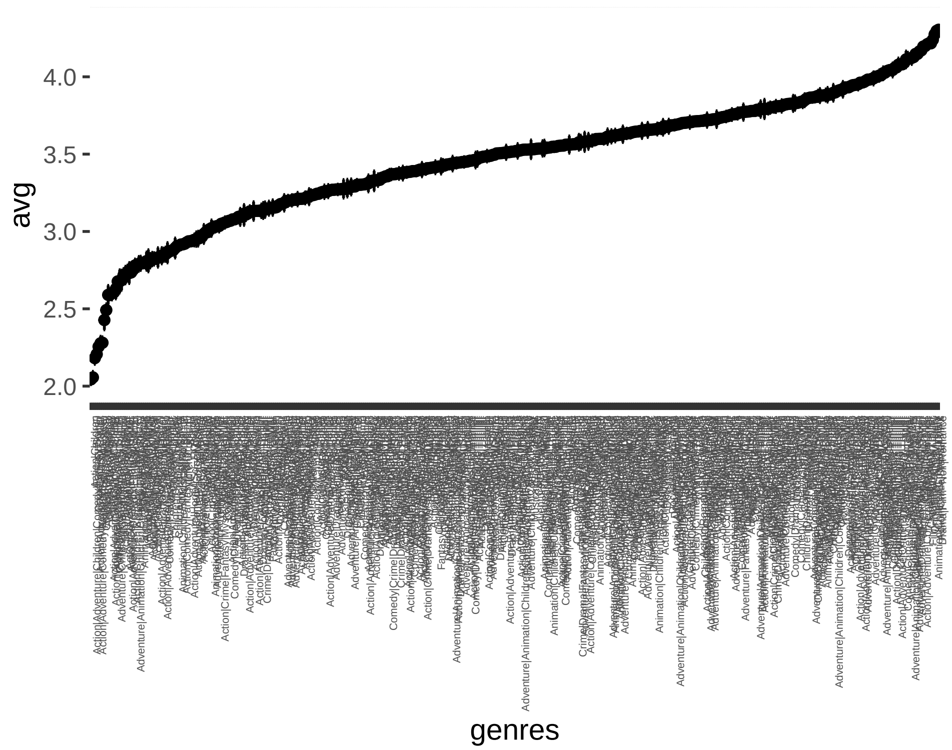
| method | RMSE |
|---|---|
| Just the average | 1.0606908 |
| Movie effect | 0.9445651 |
| Regularized Movie effect | 0.9445013 |
| Movie + User effect | 0.8669772 |
| Regularized Movie + User effect | 0.8668415 |
| Reg Movie + Reg User effect | 0.8663432 |
| Reg Movie + Reg User + Rate effect | 0.8660144 |
| Reg Movie + Reg User + Rate + Date effect | 0.8659442 |

### Genre Effects

Do action movies get better ratings on average than dramas? What about comedies? How about thrillers?

Most movies don't fit so nicely into one genre so many of the movies in the dataset have a combination of genres. Let's first see if action adventure movies differ from romance comedy movies in terms of average ratings using the `edx` dataset:

```
edx |>
  group_by(genres) |>
  summarize(n = n(),
            avg = mean(rating),
            se = sd(rating)/sqrt(n())) |>
  filter(n >= 1000) |>
  mutate(genres = reorder(genres, avg)) |>
  ggplot(aes(x = genres, y = avg, ymin = avg - 2*se, ymax = avg + 2*se)) +
  geom_point() +
  geom_errorbar() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, size = 4))
```




There does seem to be a relationship between genres and ratings, given that we have at least 1000 ratings to train on.

So let's take a look at just the genre bias effect, $b_g$, first:

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

```r
# training
genre_map <- train_set |>
  mutate(rating = rating - mu) |>
  group_by(genres) |>
  summarize(n = n(),
            b_g = mean(rating)) |>
  filter(n >= 1000) |>
  select(genres, b_g)

# test
test_set |>
  semi_join(genre_map, by = "genres") |> # keep matching genres
  left_join(genre_map, by = "genres") |> # b_g
  mutate(pred = mu + b_g) |>
  summarize(rmse = RMSE(rating, pred)) |>
  pull(rmse)
```

```
## [1] 1.019037
```

Like all of the other effects, it performs better than trivial but that's about it. Now, let's add it as another predictor to our best algorithm:

$$Y_{u,i} = \mu + b_i + b_u + b_r + b_d + b_g + \varepsilon_{u,i}$$

```r
# training
genre_to_rating <- train_set |>
  left_join(fit_movies, by = "movieId") |>
  left_join(fit_users, by = "userId") |>
  left_join(rate_map, by = "movieId") |>
  mutate(date = round_date(date, unit = "week"))
genre_map <- genre_to_rating |>
  mutate(b_r = predict(fit_rate, newdata = genre_to_rating),
         b_d = predict(fit_date, newdata = genre_to_rating),
         rating = rating - mu - b_i_reg - b_u_reg - b_r - b_d) |>
  group_by(genres) |>
  summarize(n = n(),
            b_g = mean(rating)) |>
  filter(n >= 1000) |>
  select(genres, b_g)
```

```r
# test
test_set_genre <- test_set |>
  left_join(fit_movies, by = "movieId") |>
  left_join(fit_users, by = "userId") |>
  left_join(rate_map, by = "movieId") |>
  mutate(date = round_date(date, unit = "week")) |>
  semi_join(genre_map, by = "genres") |> # keep matching genres
  left_join(genre_map, by = "genres")    # add b_g
rmse_genre <- test_set_genre |>
    mutate(b_r = predict(fit_rate, newdata = test_set_genre),
           b_d = predict(fit_date, newdata = test_set_genre),
           pred = mu + b_i_reg + b_u_reg + b_r + b_d + b_g) |>
  summarize(rmse = RMSE(rating, pred)) |>
  pull(rmse)
rmse_genre
```

```
## [1] 0.8657711
```

| method | RMSE |
|---|---:|
| Just the average | 1.0606908 |
| Movie effect | 0.9445651 |
| Regularized Movie effect | 0.9445013 |
| Movie + User effect | 0.8669772 |
| Regularized Movie + User effect | 0.8668415 |
| Reg Movie + Reg User effect | 0.8663432 |
| Reg Movie + Reg User + Rate effect | 0.8660144 |
| Reg Movie + Reg User + Rate + Date effect | 0.8659442 |
| Reg Movie + Reg User + Rate + Date + Genres effect | 0.8657711 |

Could we get even better results if we could somehow slice and dice the genres into their own categories? But first let's see what we are dealing with:

```
edx |>
  group_by(movieId) |>
  summarize(genres = first(genres)) |>
  mutate(genre_count = str_count(genres, "\\|") + 1) |>
  group_by(genre_count) |>
  summarize(movie_count = n())
```

```
## # A tibble: 8 x 2
##   genre_count movie_count
##         <dbl>       <int>
## 1           1        4004
## 2           2        3701
## 3           3        2004
## 4           4         744
## 5           5         186
## 6           6          34
## 7           7           3
## 8           8           1
```

While we have plenty of movies that fall under one genre type, the majority of them fall into multiple with some at 5+ genres. How do we train the genre bias for those? For movies with just 2 genres, does the rating effect get duplicated or split in half between them?

How about on the test set? No matter how we train the genre bias, how do we apply its effect for movies with multiple genres? Do we take the mean per genre type? Do we sum 'em all up?

We perform some exploratory analysis to see its performance:

```
# prep training for genre bias value is duplicated
genre_map <- train_set |>
  mutate(rating = rating - mu) |>
  separate_longer_delim(genres, "|") |>
  group_by(genres) |>
  summarize(n = n(),
            b_g = mean(rating)) |>
  select(genres, b_g)

# prep test for genre bias value is averaged
rmse_genre_dupe_mean <- test_set |>
  separate_longer_delim(genres, "|") |>
  left_join(genre_map, by = "genres") |> # add b_g
  mutate(pred = mu + b_g) |>
  group_by(movieId) |>
  summarize(pred = mean(pred),
            rating = first(rating)) |>
  ungroup() |>
  summarize(rmse = RMSE(rating, pred)) |>
  pull(rmse)
rmse_genre_dupe_mean
```

```
## [1] 1.150432
```

```
# prep training for genre bias value is split
genre_map <- train_set |>
```

```r
  mutate(genre_count = str_count(genres, "\\|") + 1,
         rating = (rating - mu) / genre_count) |>
  separate_longer_delim(genres, "|") |>
  group_by(genres) |>
  summarize(n = n(),
            b_g = mean(rating)) |>
  select(genres, b_g)

# prep test for genre bias value is averaged
rmse_genre_split_sum <- test_set |>
  separate_longer_delim(genres, "|") |>
  left_join(genre_map, by = "genres") |> # add b_g
  mutate(pred = mu) |>
  group_by(movieId) |>
  summarize(pred = mu + mean(b_g),
            rating = first(rating)) |>
  ungroup() |>
  summarize(rmse = RMSE(rating, pred)) |>
  pull(rmse)
rmse_genre_split_sum
```

```
## [1] 1.149649
```

```r
# prep test for genre bias value is summed
rmse_genre_split_sum <- test_set |>
  separate_longer_delim(genres, "|") |>
  left_join(genre_map, by = "genres") |> # add b_g
  mutate(pred = mu) |>
  group_by(movieId) |>
  summarize(pred = mu + sum(b_g),
            rating = first(rating)) |>
  ungroup() |>
  summarize(rmse = RMSE(rating, pred)) |>
  pull(rmse)
rmse_genre_split_sum
```

```
## [1] 46.98815
```

None of these performed better than the trivial algorithm so we can safely discard this bias from inclusion into our final algorithm.

## Results

Our final algorithm is:

$$Y_{u,i} = \mu + b_i + b_u + b_r + b_d + b_g + \varepsilon_{u,i}$$

where $\mu$ is the mean over all ratings, $b_i$ is the regularized movie bias effect, $b_u$ is the regularized user bias effect, $b_r$ is the rate bias effect, $b_d$ is the date bias effect, $b_g$ is the genre bias effect and $\varepsilon_{i,u}$ is the independent errors sampled from the same distribution centered at zero.

We are now ready to calculate the RMSE for our `final_holdout_test` set that we set aside at the beginning.

```
final_holdout_test_set <- final_holdout_test |>
  left_join(fit_movies, by = "movieId") |>
  left_join(fit_users, by = "userId") |>
  left_join(rate_map, by = "movieId") |>
  mutate(date = round_date(date, unit = "week")) |>
  semi_join(genre_map, by = "genres") |>
  left_join(genre_map, by = "genres")
nrow(final_holdout_test)
```

```
## [1] 999999
```

```
nrow(final_holdout_test_set)
```

```
## [1] 193630
```

```
rmse_final_holdout_test <- final_holdout_test_set |>
  mutate(b_r = predict(fit_rate, final_holdout_test_set),
         b_d = predict(fit_date, final_holdout_test_set),
         pred = mu + b_i_reg + b_u_reg + b_r + b_d + b_g) |>
  summarize(rmse = RMSE(rating, pred)) |>
  pull(rmse)
rmse_final_holdout_test
```

Unfortunately, the above doesn't work and will fail when trying to compute $b_r$. The reason being that there is a mismatch because `fit_rate` was chosen by training on a subset of `edx` and validated against a different but smaller subset of `edx`. Plus some movies and their ratings were removed if they didn't appear in both those subsets.

Another problem is the `semi_join` removed so many ratings that the `final_holdout_test` shrunk down to 19% of its original size. Ouch!

So let's re-train using the whole `edx` dataset using the previously selected $\lambda$s and train methods:

```
# mu
mu <- mean(edx$rating, na.rm = TRUE)
mu
```

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

```
# b_i_reg - lambda = 2.3
fit_movies <- edx |>
  mutate(rating = rating - mu) |>
  group_by(movieId) |>
  summarize(b_i_reg = sum(rating) / (n() + 2.3))

# b_u_reg - lambda = 4.8
fit_users <- edx |>
```

```r
  left_join(fit_movies, by = "movieId") |>
  mutate(rating = rating - mu - b_i_reg) |>
  group_by(userId) |>
  summarize(b_u_reg = sum(rating) / (n() + 4.8))

# b_r - knn
rate_to_rating <- edx |>
  left_join(fit_movies, by = "movieId") |>
  left_join(fit_users, by = "userId") |>
  mutate(rating = rating - mu - b_i_reg - b_u_reg) |>
  group_by(movieId) |>
  summarize(rating = mean(rating),
            rate = n() / (2023 - first(year)))
rate_map <- rate_to_rating |>
  select(movieId, rate)
fit_rate <- train(rating ~ rate, method = "knn", data = rate_to_rating)

# b_d - knn
date_to_rating <- edx |>
  left_join(fit_movies, by = "movieId") |>
  left_join(fit_users, by = "userId") |>
  left_join(rate_map, by = "movieId")
date_to_rating <- date_to_rating |>
  mutate(b_r = predict(fit_rate, date_to_rating),
         rating = rating - mu - b_i_reg - b_u_reg - b_r,
         date = round_date(date, unit = "week")) |>
  group_by(date) |>
  summarize(rating = mean(rating))
fit_date <- train(rating ~ date, method = "knn", data = date_to_rating)

# b_g
genre_to_rating <- edx |>
  left_join(fit_movies, by = "movieId") |>
  left_join(fit_users, by = "userId") |>
  left_join(rate_map, by = "movieId") |>
  mutate(date = round_date(date, unit = "week"))
genre_map <- genre_to_rating |>
  mutate(b_r = predict(fit_rate, genre_to_rating),
         b_d = predict(fit_date, genre_to_rating),
         rating = rating - mu - b_i_reg - b_u_reg - b_r - b_d) |>
  group_by(genres) |>
  summarize(n = n(),
            b_g = mean(rating)) |>
  filter(n >= 1000) |>
  select(genres, b_g)
```

And now we are finally ready to calculate the RMSE:

```
final_holdout_test_set <- final_holdout_test |>
  left_join(fit_movies, by = "movieId") |>
  left_join(fit_users, by = "userId") |>
  left_join(rate_map, by = "movieId") |>
  mutate(date = round_date(date, unit = "week")) |>
  semi_join(genre_map, by = "genres") |>
  left_join(genre_map, by = "genres")
nrow(final_holdout_test)
```

```
## [1] 999999
```

```
nrow(final_holdout_test_set)
```

```
## [1] 990560
```

```
rmse_final_holdout_test <- final_holdout_test_set |>
  mutate(b_r = predict(fit_rate, final_holdout_test_set),
         b_d = predict(fit_date, final_holdout_test_set),
         pred = mu + b_i_reg + b_u_reg + b_r + b_d + b_g) |>
  summarize(rmse = RMSE(rating, pred)) |>
  pull(rmse)
rmse_final_holdout_test
```

```
## [1] 0.8642015
```

| method | RMSE |
|---|---|
| Just the average | 1.0606908 |
| Movie effect | 0.9445651 |
| Regularized Movie effect | 0.9445013 |
| Movie + User effect | 0.8669772 |
| Regularized Movie + User effect | 0.8668415 |
| Reg Movie + Reg User effect | 0.8663432 |
| Reg Movie + Reg User + Rate effect | 0.8660144 |
| Reg Movie + Reg User + Rate + Date effect | 0.8659442 |
| Reg Movie + Reg User + Rate + Date + Genres effect | 0.8657711 |
| FINAL HOLDOUT TEST | 0.8642015 |

## Conclusion

During the research and analysis phase of this project, I was able to really leverage the knowledge I gained from taking the edX courses. Being able to fully understand and appreciate the care needed for splitting off the `final_holdout_test` so early in the process, splitting the remaining `edx` dataset into training and test sets for calculating ongoing RMSE values, and watching the RMSE values slowly fall towards zero as I added additional effective predictors was a very grueling but rewarding journey.

When looking for predictors, I was quite surprised that so many were so useful given the limited amount of variables available: `userId`, `movieId`, `rating`, `date`, `title`, `year`, `genres`. However, towards the end, I did take a path that lead to a dead end. Thankfully, most predictors kept the downward trend of RMSE values.

In the future, I can imagine that we'd be able to discover a few more effective predictors by looking for patterns in the data and even applying a matrix factorization.