Movielens Capstone Project

Philip Kassel

4/26/2021

Introduction

The purpose of this project is to improve upon the techniques described in the Movie Recommendations section of the textbook Introduction to Data Science. Using actual moving ratings from the MovieLens 10M Dataset, we want to design an algorithm that can predict the ratings that specific users will give to specific movies. To prevent overtraining of our model, the MovieLens data set will be split into two groups: a training set which is used to train and optimize our model and a validation set.

The training set contains approximately 9 million ratings and the validation set contains approximately 1 million. Both the training set and the validation set contain the actual ratings that were given to movies by specific users. The validation set, however, contains movie/user combinations that have not been seen by our model before. We can therefore compare our predicted ratings in the validation set to the true ratings in order to measure how well our model performs.

To measure the performance of our model, we will calculate the residual mean squared error (RMSE) of our predictions. RMSE can be interpreted as the average error we make when predicting a rating. The smaller the number, the better our model performs. If the RMSE is larger than 1, it means that the average prediction is more than 1 star away from the true rating, which is not good. For this project, we will aim to achieve an RMSE less than 0.86490.

Preparing the Data

Exploring the MovieLens dataset reveals that the following rating attributes can be used to train our model: the user that gave the rating, the movie that was rated, the true rating that was given, the timestamp of when the rating occurred, and the genres that were assigned to the rated movie.

```
edx[sample(.N, 5)]
```

```
##    userId movieId rating  timestamp                             title
## 1:  50805    1095    4.0  952366935       Glengarry Glen Ross (1992)
## 2:  42011    6303    3.5 1121891132     Andromeda Strain, The (1971)
## 3:  45985     593    4.0  854490846 Silence of the Lambs, The (1991)
## 4:  19660    4210    4.0 1207988321                 Manhunter (1986)
## 5:  46073     186    4.0 1123689113               Nine Months (1995)
##                                genres
## 1:                              Drama
## 2:                     Mystery|Sci-Fi
## 3:              Crime|Horror|Thriller
## 4: Action|Crime|Drama|Horror|Thriller
## 5:                     Comedy|Romance
```

Included in the title of each rated movie is the year that the movie was released, which we will extract because it might be a useful predictor. We will also measure the length of time between each movie's release date and when each rating occurred, rounded the nearest 5 year increment.

```
edx$release_year <- str_sub(edx$title,start= -6)
edx$release_year <- as.numeric(str_extract(edx$release_year, "\\d+"))
edx$rating_year <- year(as.Date(as.POSIXct(edx$timestamp,origin="1970-01-01")))
edx$years_since_release <- round((edx$rating_year - edx$release_year) / 5) * 5

validation$release_year <- str_sub(validation$title,start= -6)
validation$release_year <- as.numeric(str_extract(validation$release_year, "\\d+"))
validation$rating_year <- year(as.Date(as.POSIXct(validation$timestamp,origin="1970-01-01")))
# Round to the nearest 10 year increment
validation$years_since_release <- round((validation$rating_year - validation$release_year) / 5) * 5

edx[sample(.N, 5)]
```

```
##    userId movieId rating  timestamp
## 1:  13268     490    4.0  923662715
## 2:   6910     720    0.5 1190501372
## 3:  30265    2408    2.0  974969711
## 4:  65174     930    3.5 1159822942
## 5:  18227    6240    5.0 1050522610
##                                                   title
## 1:                                       Malice (1993)
## 2: Wallace & Gromit: The Best of Aardman Animation (1996)
## 3:                           Cocoon: The Return (1988)
## 4:                                    Notorious (1946)
## 5:                                 One Good Cop (1991)
##                       genres release_year rating_year years_since_release
## 1:                  Thriller         1993        1999                   5
## 2: Adventure|Animation|Comedy         1996        2007                  10
## 3:             Comedy|Sci-Fi         1988        2000                  10
## 4: Film-Noir|Romance|Thriller         1946        2006                  60
## 5:         Action|Crime|Drama         1991        2003                  10
```

Techniques

To train our model, we will first establish a baseline, *mu*, which is simply the average rating given in our training set. We will then build on this baseline by exploring how the following effects can influence a specific user's rating of a movie: The movie itself, the user that rated the movie, the user's genre preferences, and the time that elapsed between the release of the movie and when the rating was given.

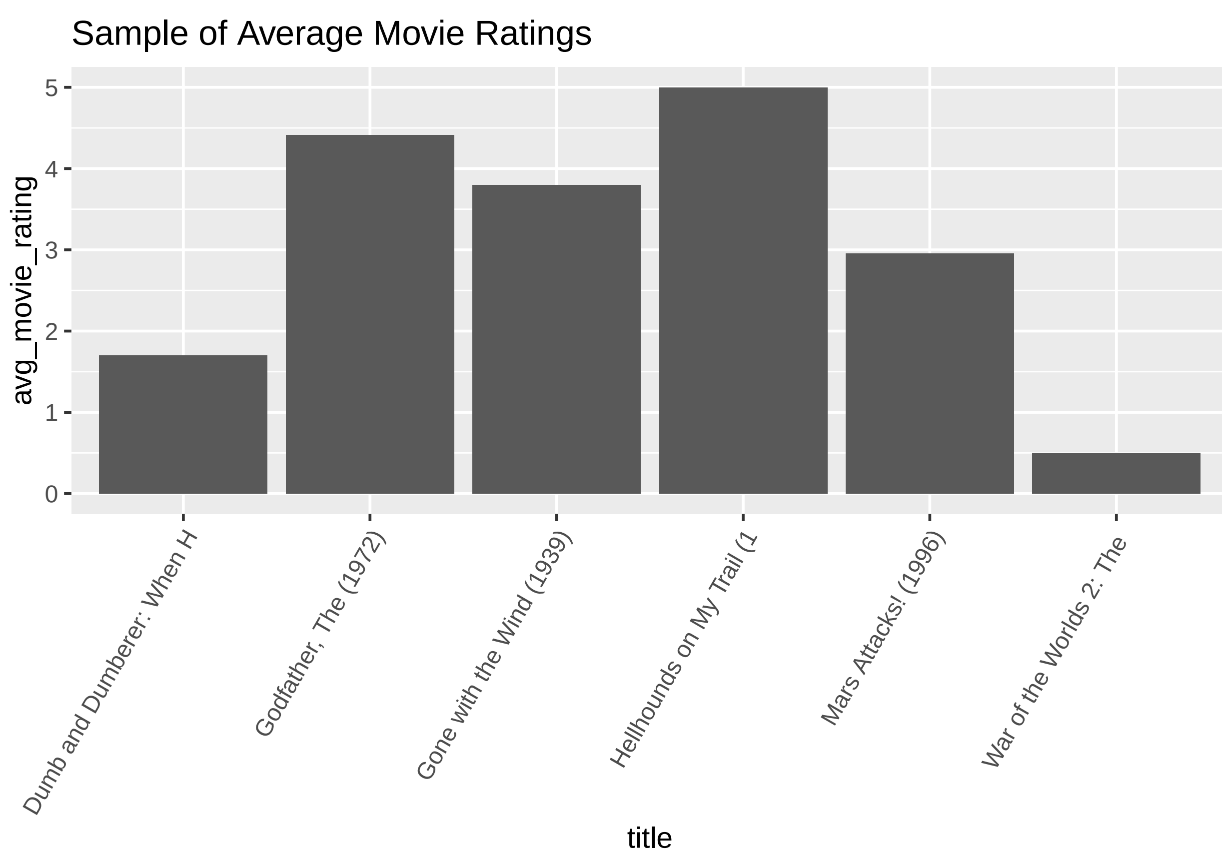
```
mu <- mean(edx$rating)
paste("Average Movie Rating:", round(mu, 3))
```

```
## [1] "Average Movie Rating: 3.512"
```

Movie Effect

Now that we have our baseline, we can approximate how different effects can influence the way that a particular rating will deviate from the average rating. For example, we know that some movies are more highly regarded than others, so we would expect for those movies to be rated higher than average. Some movies are also known flops, so we would except for those movies to be rated lower than average.

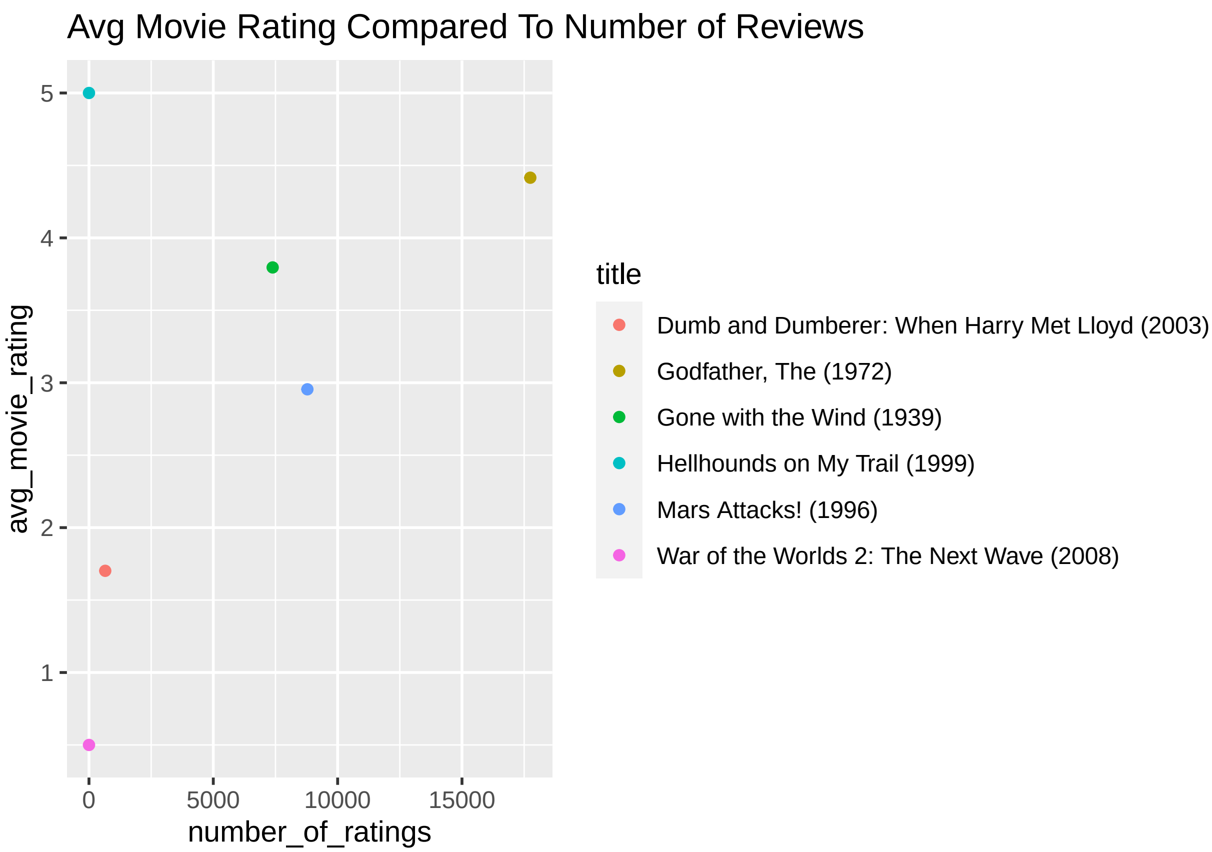
```
titles <- c("Mars Attacks! (1996)",
            "Dumb and Dumberer: When Harry Met Lloyd (2003)",
            "Godfather, The (1972)",
            "Gone with the Wind (1939)",
            "War of the Worlds 2: The Next Wave (2008)",
            "Hellhounds on My Trail (1999)")

edx %>% filter(title %in% titles) %>% group_by(title) %>%
  summarise(avg_movie_rating=mean(rating)) %>% arrange(avg_movie_rating) %>%
  mutate(title = strtrim(title, 25)) %>% ggplot(aes(title, avg_movie_rating)) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle = 60, hjust = 1)) +
  ggtitle("Sample of Average Movie Ratings")
```



Some of these movies we would expect to have such high or low ratings; for example, the sequel to "Dumb and Dumber", "The Godfather", and "Gone with the Wind". But, is it really likely that the average user will rate "Hellhounds on My Trail" higher than "The Godfather"? As it turns out, the less often a movie is rated, the more likely it is to appear at the extremes of the rating spectrum. In the following visual, we can clearly see that the movies with the highest and lowest ratings have been rated very few times.

```
edx %>% filter(title %in% titles) %>% group_by(title) %>%
  summarise(avg_movie_rating=mean(rating), number_of_ratings = n()) %>%
  arrange(number_of_ratings) %>%
```

```r
ggplot(aes(number_of_ratings, avg_movie_rating, color=title)) +
geom_point() + ggtitle("Avg Movie Rating Compared To Number of Reviews")
```



Using a technique called regularization, we can apply a parameter to our ratings, which we'll call *lambda*, that will penalize underrated movies in order to better approximate how they might deviate from our average rating, *mu*. We'll tune this parameter using cross-validation, so that we can find the best value for *lambda* that minimizes our RMSE score. Once *lambda* has been tuned, we can calculate the movie bias by subtracting the regularized average rating for each movie from our baseline *mu*.

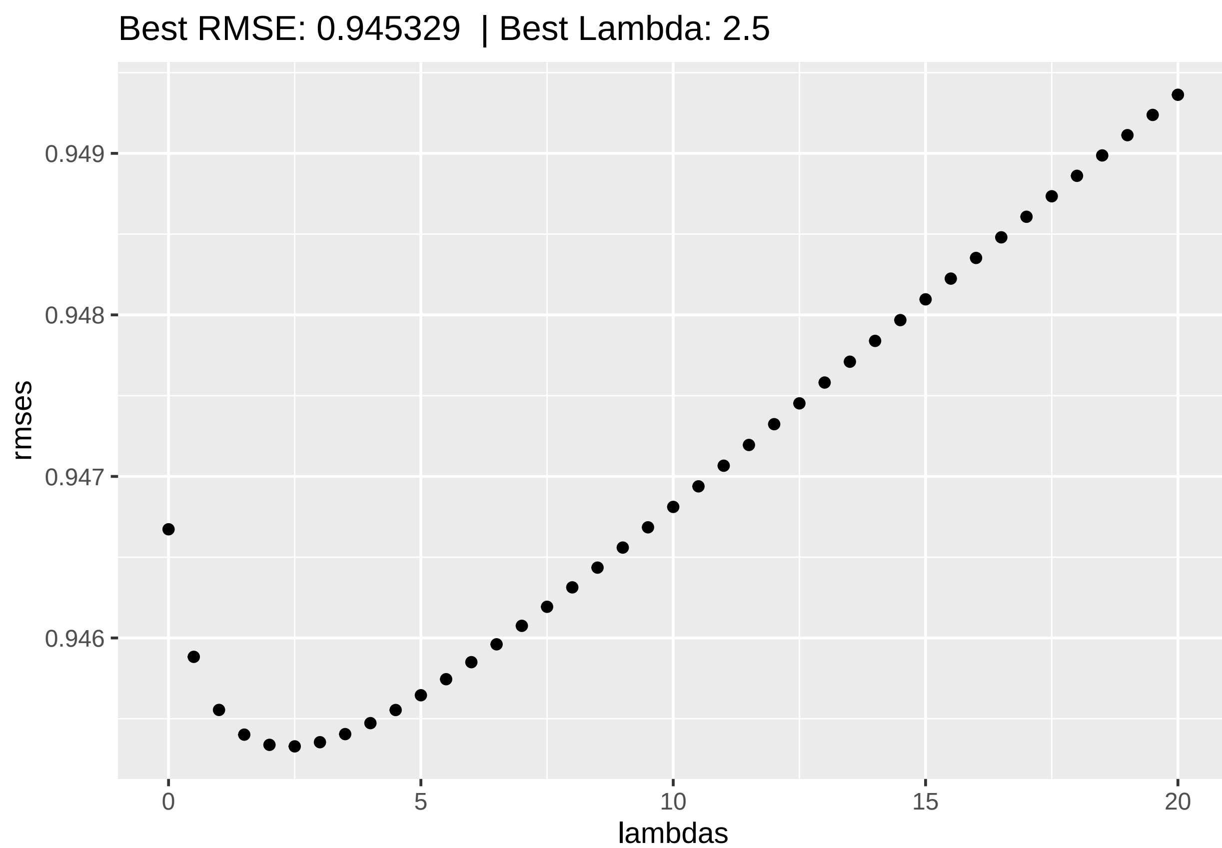
Since tuning our lambda parameter will require us to run our model many times, we will use only a sample of one million records from our training set. To prevent over training of our model (which would lead to less accurate predictions on our validation set), we will further split our training set: 90% will be used to train our tuning model and the remaining 10% will be used to test the accuracy of our predictions. The *lambda* parameter that helps us achieve the lowest residual mean error (RMSE) in our predictions will be the one that is used in our final model.

```r
### Collect a sample of 1 million ratings from the training set end separate
### it into our training and test set

sample <- edx[1:1000000]
ind <- createDataPartition(sample$rating, times = 1, p=0.1, list=FALSE)
train_set <- sample[-ind]
test_set <- sample[ind]

### we want to make sure that the test set only has movies and users that are in
### the train set
```

```r
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId") %>%
  semi_join(train_set, by = "years_since_release")


mu <- mean(train_set$rating)
lambdas <- seq(0,20,.5)

tune_lambdas <- function(grouping) {
  if (grouping == "genre") {
    biases <- genre_ratings_df %>% group_by(userId, genres)
  }

  else {
    biases <- train_set %>% group_by(!!as.symbol(grouping))
  }
  rate_lambda <- function(lambda) {
    if (grouping == "genre") {
      temp <- biases %>%
        summarise(genre_bias = sum(rating - movie_bias - user_bias - mu) /
                    (n() + lambda))
      temp <- temp %>% inner_join(test_genre_ratings_df, on=genres)

      predictions <- temp %>%
        left_join(movies_df, on="movieId") %>%
        left_join(users_df, on="userId") %>%
        mutate(pred=mu + genre_bias + movie_bias + user_bias)
    }
    else if (grouping == "userId") {
      temp <- biases %>%
        summarise(user_bias = sum(rating - movie_bias - mu)/(n() + lambda))

      predictions <- test_set %>%
        inner_join(temp, on=!!as.symbol(grouping)) %>%
        left_join(movies_df, on=movieId) %>%
        mutate(pred = mu + movie_bias + user_bias) %>%
        select(rating, pred)
    }
    else if (grouping == "years_since_release") {
      ug <- user_genres_df %>% select(-movieId) %>%
        group_by(userId, genres) %>% summarise(genre_bias = mean(genre_bias))

      tg <- test_genre_ratings_df %>% inner_join(ug, on=genres) %>%
        group_by(userId, movieId) %>%
        summarise(user_genre_bias = mean(genre_bias))

      temp <- biases %>%
        summarise(bias=sum(rating - mu - movie_bias - user_bias - user_genre_bias) / (n() + lambda))

      predictions <- test_set %>% left_join(temp, on=years_since_release) %>%
        left_join(movies_df, on=movieId) %>% left_join(users_df, on=userId) %>%
        left_join(tg, on=c(userId, movieId))
```

```r
      predictions[is.na(predictions)] = 0

      predictions <- predictions %>%
        mutate(pred = mu + movie_bias + user_bias + user_genre_bias + bias)
    }
    else {
    temp <- biases %>% summarise(bias=sum(rating - mu) / (n() + lambda))
    predictions <- test_set %>% inner_join(temp, on=!!as.symbol(grouping)) %>%
      mutate(pred = mu + bias) %>% select(rating, pred)
    }

    RMSE(predictions$rating, predictions$pred)
  }
  sapply(lambdas, FUN = function(x) rate_lambda(x))
}

movies_lambda <- tune_lambdas("movieId")
movies_df <- train_set %>% group_by(movieId) %>%
  summarise(movie_bias=sum(rating - mu)/(n() +
  lambdas[which.min(movies_lambda)])) %>%
  select(movieId, movie_bias)

train_set <- train_set %>% inner_join(movies_df, on="movieId")
```

```r
data.frame(rmses=movies_lambda, lambdas = lambdas) %>%
  ggplot(aes(lambdas, rmses)) + geom_point() +
  ggtitle(paste("Best RMSE:", round(min(movies_lambda),6),
                " | Best Lambda:", lambdas[which.min(movies_lambda)]))
```



In order to make predictions about how a user might rate an unseen movie, we can simply add the calculated *movie_bias* to our baseline average *mu* like so: *predicted rating = movie_bias + mu.*

```
train_set[sample(.N, 5)] %>% inner_join(movies_df, on=movieId) %>%
  select(title, movie_bias) %>% mutate(mu = mu) %>%
  mutate(predicted_rating = movie_bias + mu)
```

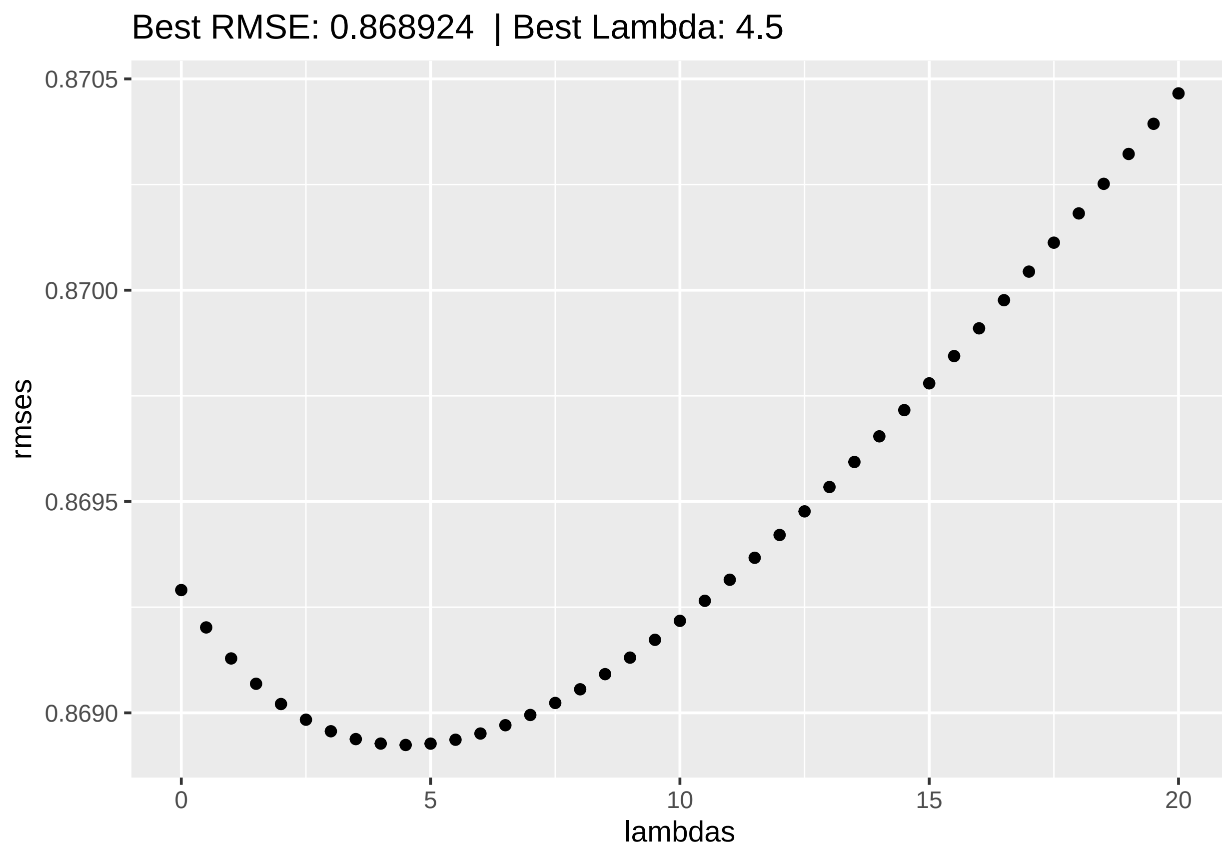
```
##                                          title   movie_bias       mu
## 1:                          Waterworld (1995) -0.642198610 3.520253
## 2:       Thing from Another World, The (1951)  0.195889823 3.520253
## 3: Life Aquatic with Steve Zissou, The (2004)  0.004688592 3.520253
## 4:                          Jack Frost (1998) -1.176453689 3.520253
## 5:                      Mrs. Doubtfire (1993) -0.093880789 3.520253
##    predicted_rating
## 1:         2.878054
## 2:         3.716143
## 3:         3.524941
## 4:         2.343799
## 5:         3.426372
```

User Effects

If not all movies are created equal, then it stands to reason that not all users are, either. For example, Jane may be a movie buff that rates movies more critically than John, who only watches blockbuster hits. We can, therefore, build upon our previous *movie effect*, by looking at each user's rating for a movie and subtracting from it that movie's *movie_bias* and our baseline *mu*.

We will then use regularization and cross validation to penalize users with fewer ratings in order to find the best value of *lambda* (and therefore calculate each user's *user bias*) that minimizes RMSE.

```
users_lambda <- tune_lambdas("userId")
users_df <- train_set %>% group_by(userId) %>%
  summarise(user_bias=sum(rating - movie_bias - mu)/(n() +
  lambdas[which.min(users_lambda)])) %>%
  select(userId, user_bias)

data.frame(rmses=users_lambda, lambdas = lambdas) %>%
  ggplot(aes(lambdas, rmses)) + geom_point() + ggtitle(paste("Best RMSE:",
    round(min(users_lambda),6), " | Best Lambda:",
    lambdas[which.min(users_lambda)]))
```




```
train_set <- train_set %>% inner_join(users_df, on="userId")
```

Already, we can see a significant improvement in our predictions, as measured by RMSE.

With our new approximation, we can predict how a user will rate an unseen movie by adding their known *user_bias* to the particular movie's known *movie_bias* and adding that to our baseline *mu*.

In other words: *predicted rating = user_bias + movie_bias + mu*.

```
train_set[sample(.N, 5)] %>% inner_join(movies_df, on=movieId) %>%
  inner_join(users_df, on=userId) %>%
  select(title, movie_bias, user_bias) %>% mutate(mu = mu) %>%
  mutate(predicted_rating = movie_bias + user_bias + mu)
```

```
##                                     title movie_bias  user_bias       mu
## 1:                      Toy Story 2 (1999)  0.3250506  0.2409186 3.520253
## 2:            Sense and Sensibility (1995)  0.4864494  0.2056762 3.520253
## 3:      Searching for Bobby Fischer (1993)  0.3929038  0.2294676 3.520253
## 4: Nightmare Before Christmas, The (1993)  0.1894709  0.6201546 3.520253
## 5:                           Casino (1995)  0.1748299 -0.1563618 3.520253
##    predicted_rating
## 1:         4.086222
## 2:         4.212378
## 3:         4.142624
## 4:         4.329878
## 5:         3.538721
```

User Genres Effect

Some users prefer certain genres more than others. This effect is more challenging to approximate because many movies can be categorized into multiple genres.

```
##    userId movieId rating  timestamp                             title
## 1:    786     258    1.0  913064925 Kid in King Arthur's Court, A (1995)
## 2:   7289     663    3.0  853321394 Kids in the Hall: Brain Candy (1996)
## 3:   7070     208    3.0  839507492                    Waterworld (1995)
## 4:   5671     357    2.5 1169369270   Four Weddings and a Funeral (1994)
## 5:   6776     377    0.5 1190536559                         Speed (1994)
##                                   genres release_year rating_year
## 1: Adventure|Children|Comedy|Fantasy|Romance         1995        1998
## 2:                                  Comedy         1996        1997
## 3:                  Action|Adventure|Sci-Fi         1995        1996
## 4:                          Comedy|Romance         1994        2007
## 5:                  Action|Romance|Thriller         1994        2007
##    years_since_release  movie_bias  user_bias
## 1:                   5 -0.99042085 -0.1098489
## 2:                   0 -0.14863137 -0.3903074
## 3:                   0 -0.64219861 -0.3825903
## 4:                  15  0.14720162 -0.6931864
## 5:                  15 -0.03463044  0.1440741
```

To measure a user's genre preferences, we will need to split each of their ratings up by their constituent genres. Below, we can see the sample ratings above broken down by genre.

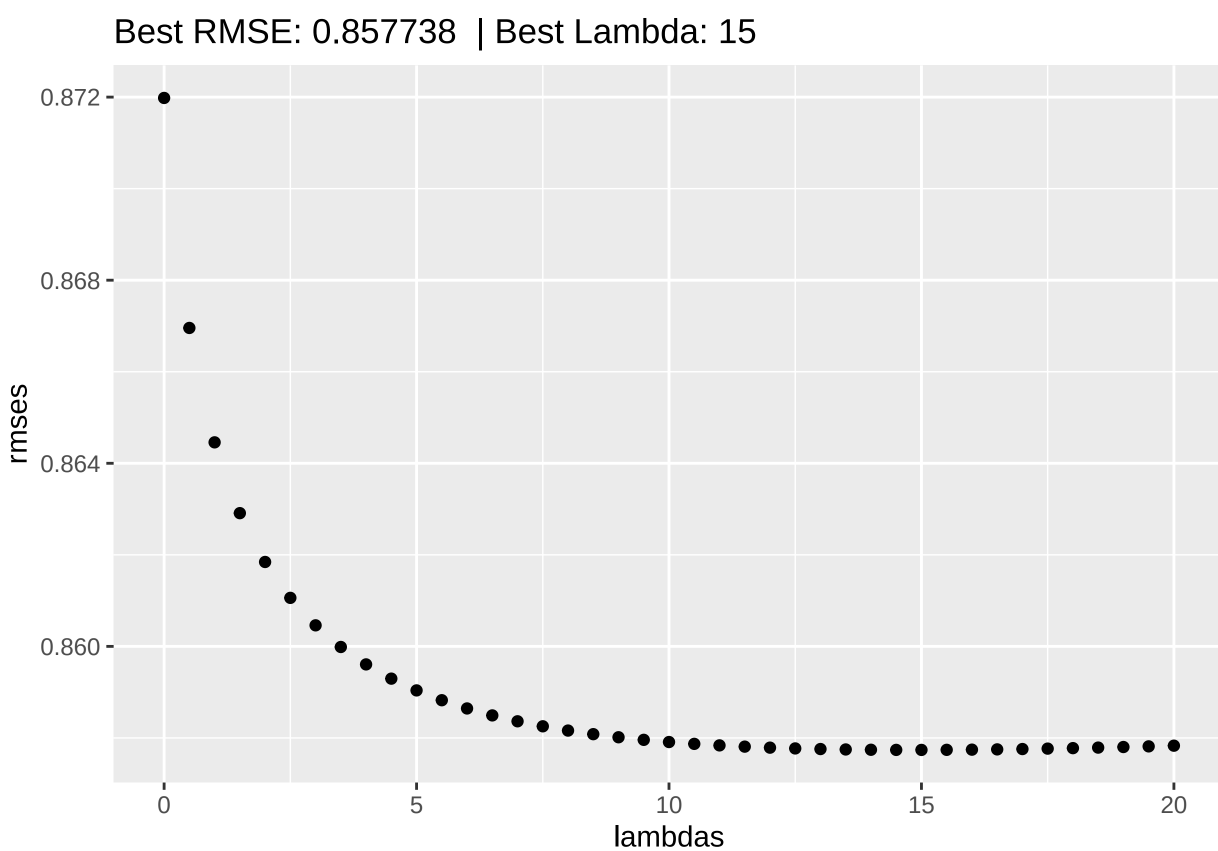
```
g %>% separate_rows(genres, sep="\\|") %>% select(userId, rating, title, genres)
```

```
## # A tibble: 14 x 4
##    userId rating title                                genres
##     <int>  <dbl> <chr>                                <chr>
##  1    786      1 Kid in King Arthur's Court, A (1995) Adventure
##  2    786      1 Kid in King Arthur's Court, A (1995) Children
##  3    786      1 Kid in King Arthur's Court, A (1995) Comedy
##  4    786      1 Kid in King Arthur's Court, A (1995) Fantasy
##  5    786      1 Kid in King Arthur's Court, A (1995) Romance
##  6   7289      3 Kids in the Hall: Brain Candy (1996) Comedy
##  7   7070      3 Waterworld (1995)                    Action
```

```
## 8   7070   3   Waterworld (1995)                    Adventure
## 9   7070   3   Waterworld (1995)                    Sci-Fi
## 10  5671   2.5 Four Weddings and a Funeral (1994)   Comedy
## 11  5671   2.5 Four Weddings and a Funeral (1994)   Romance
## 12  6776   0.5 Speed (1994)                         Action
## 13  6776   0.5 Speed (1994)                         Romance
## 14  6776   0.5 Speed (1994)                         Thriller
```

After this is done, we can use regularization and cross validation to calculate a specific bias that *each user has for each genre they have rated.*

```r
genre_ratings_df <- train_set %>% separate_rows(genres, sep="\\|")
test_genre_ratings_df <- test_set %>% separate_rows(genres, sep="\\|")
genres_lambda <- tune_lambdas("genre")

data.frame(rmses=genres_lambda, lambdas = lambdas) %>%
  ggplot(aes(lambdas, rmses)) + geom_point() +
  ggtitle(paste("Best RMSE:", round(min(genres_lambda),6),
                " | Best Lambda:", lambdas[which.min(genres_lambda)]))
```



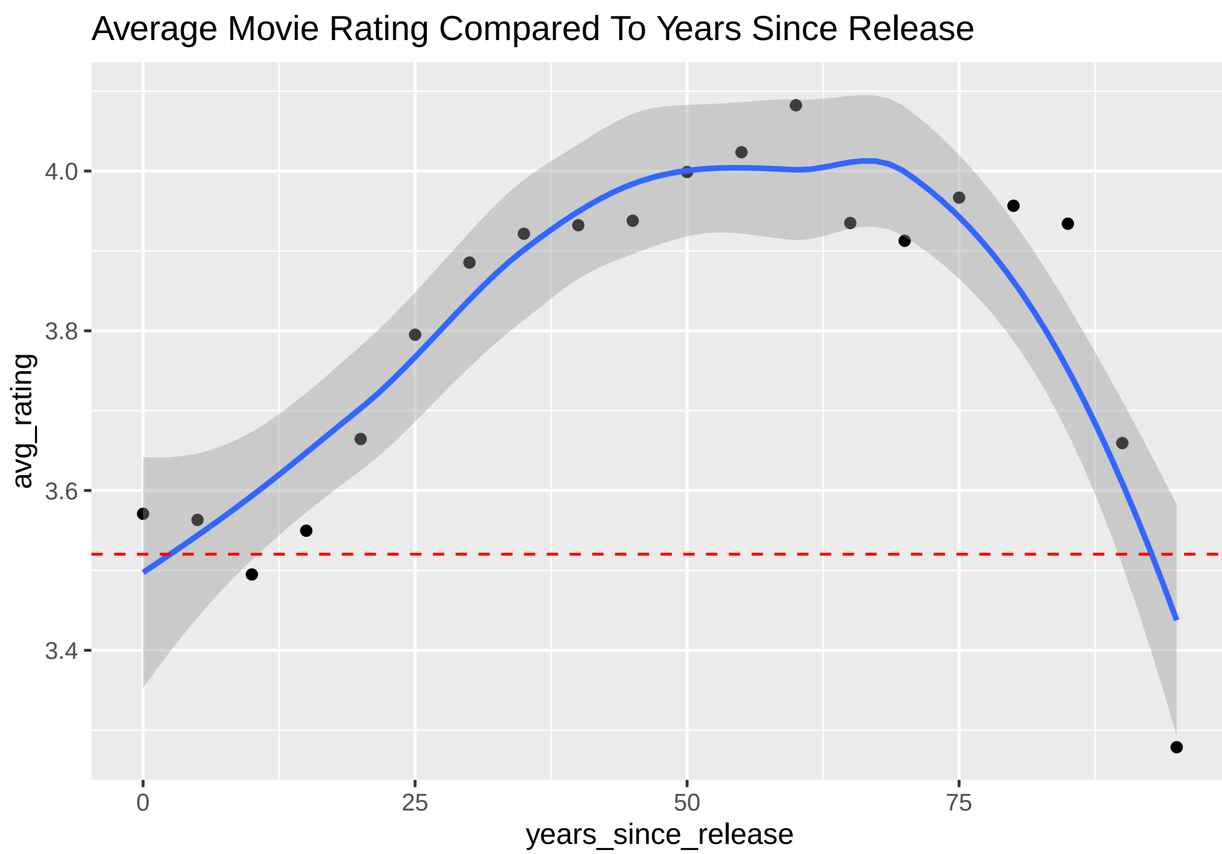
Once we have measured each user's bias for every genre, we can then assign those genre biases to the movies in our validation set. Since one movie may have multiple genres and each genre bias was measured separately, we will calculate the average of all genre biases for all movie/user/genre combinations, which we'll call *user_genre_bias*.

```r
user_genres_df <- genre_ratings_df %>% group_by(userId, genres) %>%
  summarise(genre_bias = sum(rating - movie_bias - user_bias - mu)/(n() +
  lambdas[which.min(genres_lambda)])) %>%
  inner_join(genre_ratings_df, on=genres) %>%
  select(userId, genres, genre_bias, movieId)


user_genres_df_wide <- user_genres_df %>%
  pivot_wider(names_from = genres, values_from=genre_bias)

train_set <- train_set %>%
  inner_join(user_genres_df_wide, on=c("userId", "movieId"))
```

```
##    userId rating                         title rating_year Action Adventure
## 1:   3174    3.5             Baby Mama (2008)        2008     NA        NA
## 2:   6245    2.0 Little Shop of Horrors (1986)        2001     NA        NA
## 3:   4383    4.0             Boomerang (1992)        1996     NA        NA
## 4:   5208    3.0      Conspiracy Theory (1997)        2001     NA        NA
## 5:    821    3.0       Sixth Sense, The (1999)        2005     NA        NA
##    Animation Children     Comedy Crime       Drama Fantasy   Musical    Romance
## 1:        NA       NA 0.09165471    NA          NA      NA        NA         NA
## 2:        NA       NA 0.05462930    NA          NA      NA 0.1653844         NA
## 3:        NA       NA 0.09008663    NA          NA      NA        NA  0.1197351
## 4:        NA       NA         NA    NA -0.05241881      NA        NA -0.1570873
## 5:        NA       NA         NA    NA  0.17493261      NA        NA         NA
##    Sci-Fi   Thriller War     Mystery Western Film-Noir     Horror Documentary
## 1:     NA         NA  NA          NA      NA        NA         NA          NA
## 2:     NA         NA  NA          NA      NA        NA -0.2737409          NA
## 3:     NA         NA  NA          NA      NA        NA         NA          NA
## 4:     NA  0.1055971  NA -0.21598924      NA        NA         NA          NA
## 5:     NA -0.2743949  NA -0.02007429      NA        NA         NA          NA
##    IMAX (no genres listed) user_genre_bias
## 1:   NA                 NA      0.09165471
## 2:   NA                 NA     -0.01790907
## 3:   NA                 NA      0.10491085
## 4:   NA                 NA     -0.07997456
## 5:   NA                 NA     -0.03984552
```

Time Effect

When it comes to movies, there are warhorses that stand the test of time; classics like "Gone with the Wind" and "Citizen Kane". Indeed, we can see a fairly strong correlation between the average rating and the length of time since a movie was released.

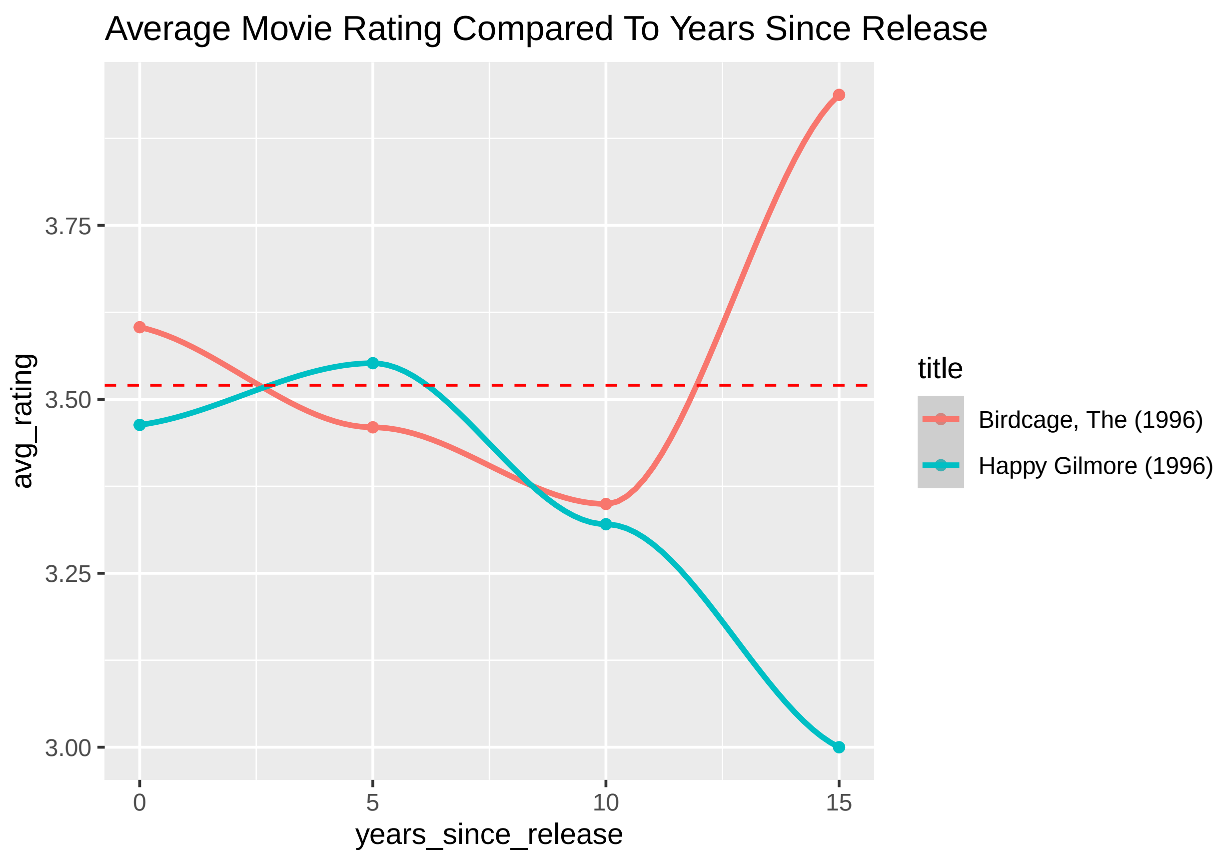


We can see clearly that movies are rated slightly higher than average when they are first released and then take a dip within the first 10 years or so. After about 25 years, movies are rated increasingly rated higher. Perhaps movies that were made longer ago are sought out specifically because they are more critically acclaimed. No matter the reason, this appears to be a useful predictor to include in our model.

We can also see that not every movie ages as well. Therefore, we will we measure the effect that time can have on the average rating for every movie in our dataset. We will call this final parameter *recency_bias*.

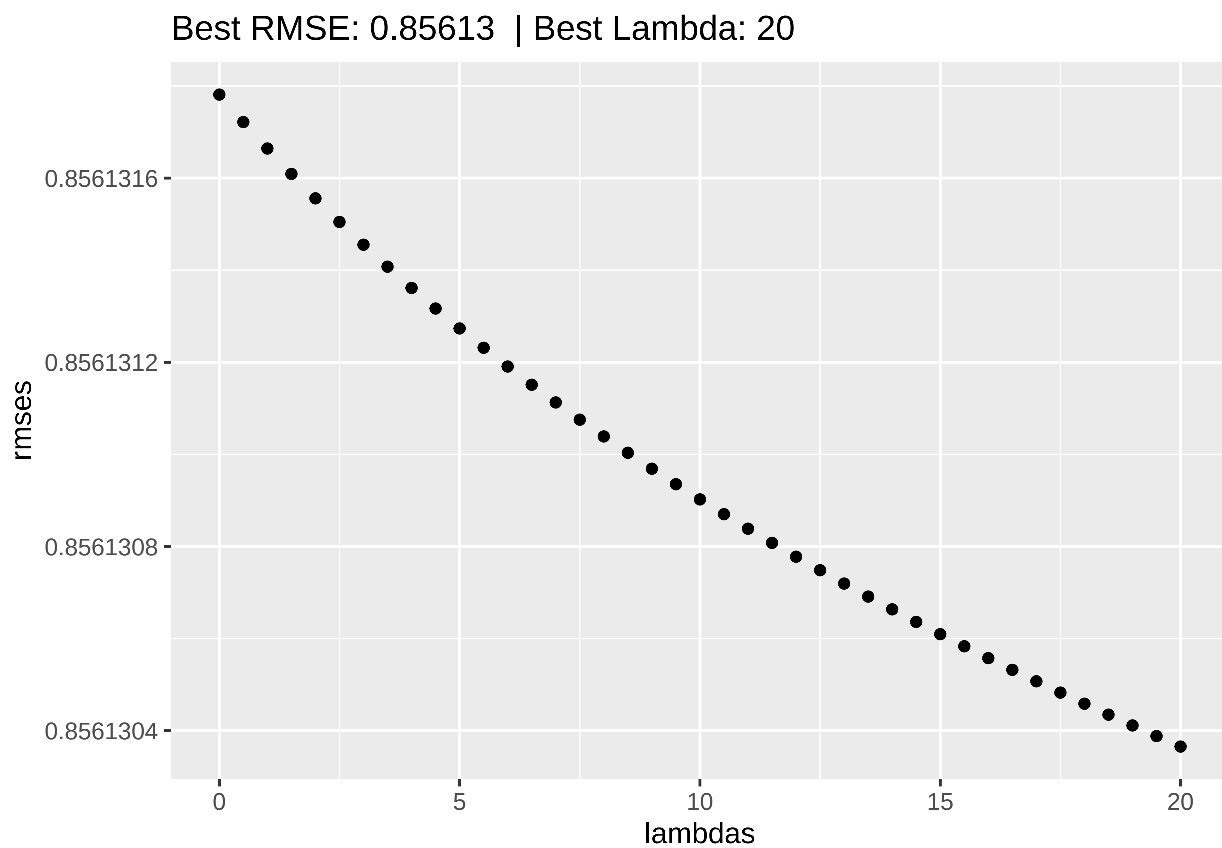
```
edx %>% filter(title %in% c("Birdcage, The (1996)", "Happy Gilmore (1996)")) %>%
  group_by(years_since_release, title) %>% summarise(avg_rating=mean(rating)) %>%
  ggplot(aes(years_since_release, avg_rating, color=title)) +
  geom_point() + geom_smooth() +
  geom_hline(yintercept=mu, linetype="dashed", color = "red", size=.5) +
  ggtitle("Average Movie Rating Compared To Years Since Release")
```

```
## `summarise()` has grouped output by 'years_since_release'. You can override using the `.groups` argu

## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



As we did with our previous models, we will use regularization and cross validation to tune our penalty term *lambda* to minimize our RMSE.

```
years_since_release_lambda <- tune_lambdas("years_since_release")
recency_df <- train_set %>% group_by(years_since_release) %>%
  summarise(recency_bias=sum(rating - movie_bias -
  user_bias - user_genre_bias - mu)/(n() +
  lambdas[which.min(years_since_release_lambda)])) %>%
  select(years_since_release, recency_bias)

train_set <- train_set %>% inner_join(recency_df, on="years_since_release")

data.frame(rmses=years_since_release_lambda, lambdas = lambdas) %>%
  ggplot(aes(lambdas, rmses)) + geom_point() +
  ggtitle(paste("Best RMSE:", round(min(years_since_release_lambda),6),
  " | Best Lambda:", lambdas[which.min(years_since_release_lambda)]))
```



Each additional bias improves our score, but we seem to be approaching the limit. Now that we've tuned our lambda parameters, we can combine our different effects and test our model on our validation set. The lambda parameters to that will be used for each effect are:

```
## [1] "Movie Lambda: 2.5"      "Users Lambda: 4.5"      "User Genres Lambda: 15"
## [4] "Recency Lambda: 20"
```

These four optimal lambda parameters will now be used to separately calculate movie, user, user-genre, and recency biases for our entire training set. We will then use those biases to make ratings predictions for the unseen movie ratings in our validation set. The formula for our prediction is simply:

predicted rating = user_bias + movie_bias + user_genre_bias + recency_bias + mu.

Conclusion

Final RMSE (Validation Set): 0.85355

To showcase the performance of our model, we can observe a random selection of ratings from the validation set and compare our predicted rating against the true rating that was given.

```
##     userId                                  title rating prediction
##  1:  53607                  Lady in the Water (2006)    1.0        3.0
##  2:  12239            Farinelli: il castrato (1994)    3.5        3.5
##  3:  59659                         Blind Date (1987)    2.0        2.0
##  4:  15567       Star Trek: The Motion Picture (1979)    3.0        3.0
```

```
##  5:  67615                             Face/Off (1997)    2.0        3.0
##  6:  29979                     Starship Troopers (1997)    3.0        3.5
##  7:  14145 Willy Wonka & the Chocolate Factory (1971)    3.0        3.5
##  8:  71055                           Parenthood (1989)    3.0        3.0
##  9:  19284   Ghost Dog: The Way of the Samurai (1999)    4.5        4.0
## 10:   3035            Silence of the Lambs, The (1991)    5.0        5.0
## 11:  55707                 Beauty and the Beast (1991)    3.0        3.5
## 12:  52307                  Saving Private Ryan (1998)    5.0        4.5
## 13:  18602                            King Kong (1976)    3.5        3.5
## 14:  19564                          Dante's Peak (1997)    5.0        3.0
## 15:  46694                   Married to the Mob (1988)    3.0        3.0
## 16:  35537  Police Academy 3: Back in Training (1986)    3.0        2.0
## 17:  66167                           Others, The (2001)    4.0        3.5
## 18:  16487                         Roman Holiday (1953)    5.0        3.5
## 19:  46732                           Blown Away (1994)    4.0        3.5
## 20:  66387                         Fugitive, The (1993)    4.0        4.0
```

Overall, the model performs very well against the validation set and with relatively few predictors. While we were successful in achieving our goal of a residual mean error (RMSE) < 0.86490, the author would like to point out that, even when only applying cross-validation on approximately 10% of the training set, it takes a very long time for the model to run. It likely will not scale well against larger datasets.

Having to calculate the *user_genre_bias* for every user/genre combination by breaking the training and validation sets into their individual genres only to reassemble them back into a single movie rating likely contributed the most to the slow performance of this model. While it was very effective in reducing RMSE, refactoring the way that this bias is calculated could very well improve the scalability of our model.