Predicting Movie Recommendations

Sohum Sanghvi

3/6/2020

Introduction

The MovieLens dataset contains roughly 10 million ratings of movies, provided by 100,000 users. The goal of this project is to predict the ratings that a user will give to movies they have not yet rated. We start by exploring the variables in the dataset, such as the distribution of the ratings and which movies have been rated the most. After the initial analysis, we start building a dataset to predict ratings for. We create a dataset of users with movies they have not rated, and through machine learning approaches, we aim to predict the ratings the users will provide.

Analysis

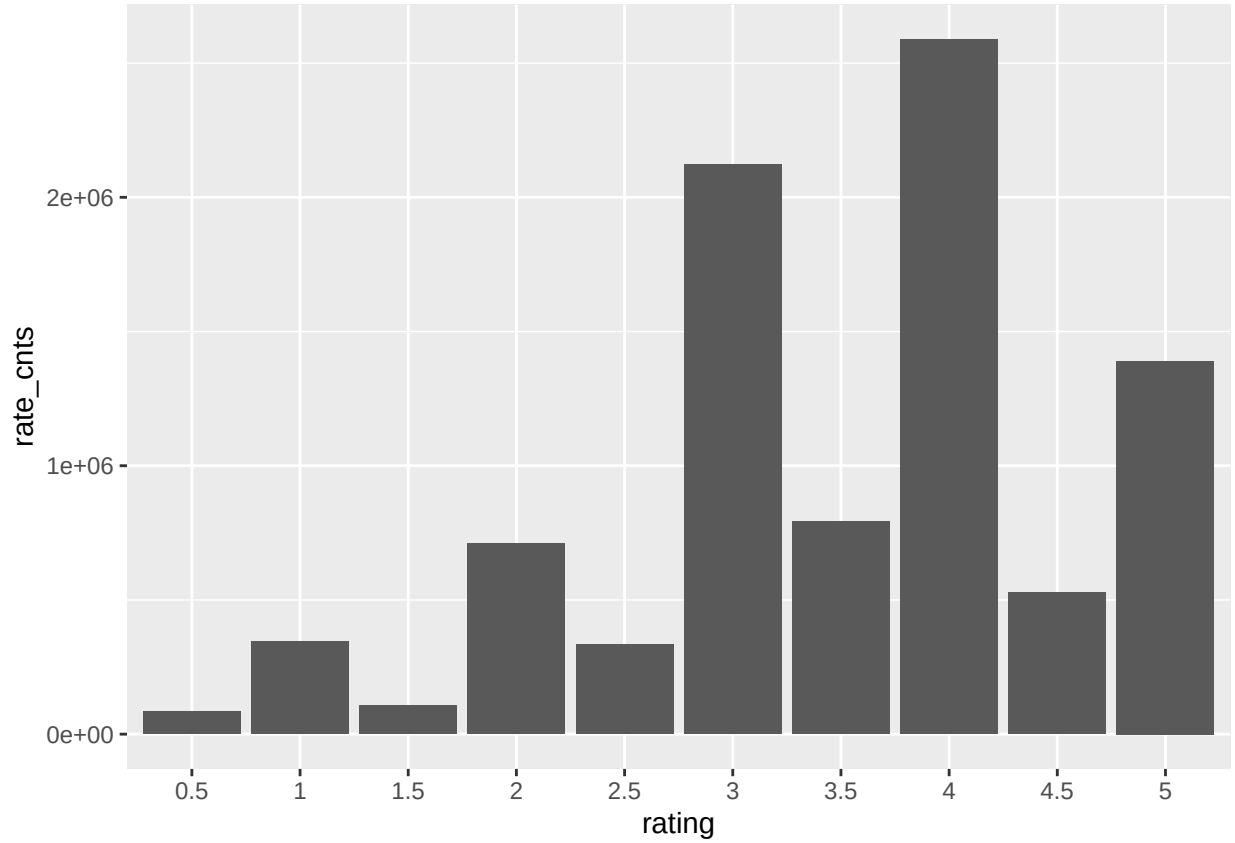
We begin by examining the edx (training) dataset. We can get a summary of the data as follows:

```
summary(edx)
```

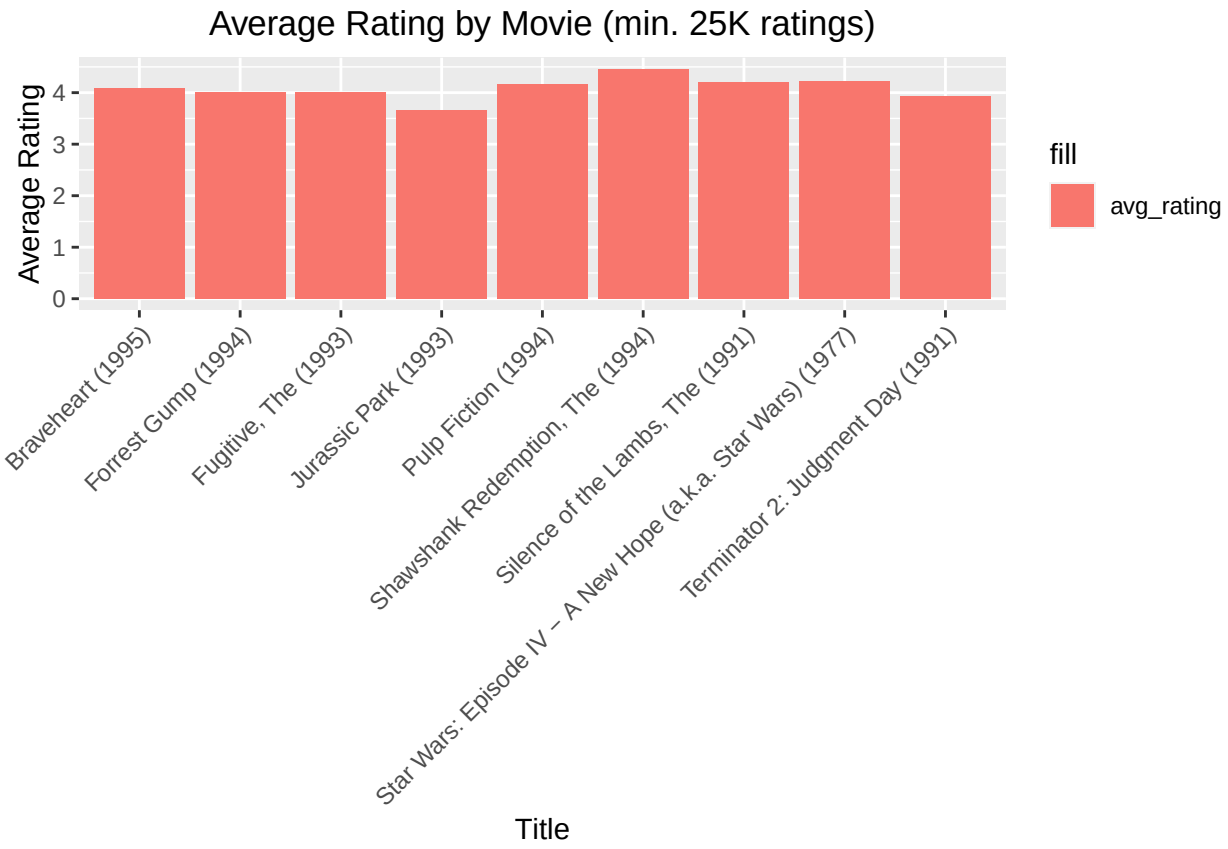
```
##      userId         movieId          rating        timestamp
##  Min.   :    1   Min.   :    1   Min.   :0.500   Min.   :7.897e+08
##  1st Qu.:18124   1st Qu.:  648   1st Qu.:3.000   1st Qu.:9.468e+08
##  Median :35738   Median : 1834   Median :4.000   Median :1.035e+09
##  Mean   :35870   Mean   : 4122   Mean   :3.512   Mean   :1.033e+09
##  3rd Qu.:53607   3rd Qu.: 3626   3rd Qu.:4.000   3rd Qu.:1.127e+09
##  Max.   :71567   Max.   :65133   Max.   :5.000   Max.   :1.231e+09
##     title              genres
##  Length:9000055     Length:9000055
##  Class :character   Class :character
##  Mode  :character   Mode  :character
##
##
##
```

We can start analyzing the distribution of ratings. From the summary, we know the ratings are always between 0.5 and 5 (inclusive), the average rating is about 3.5, and the median rating is 4. We further break down the average ratings by genre and year.

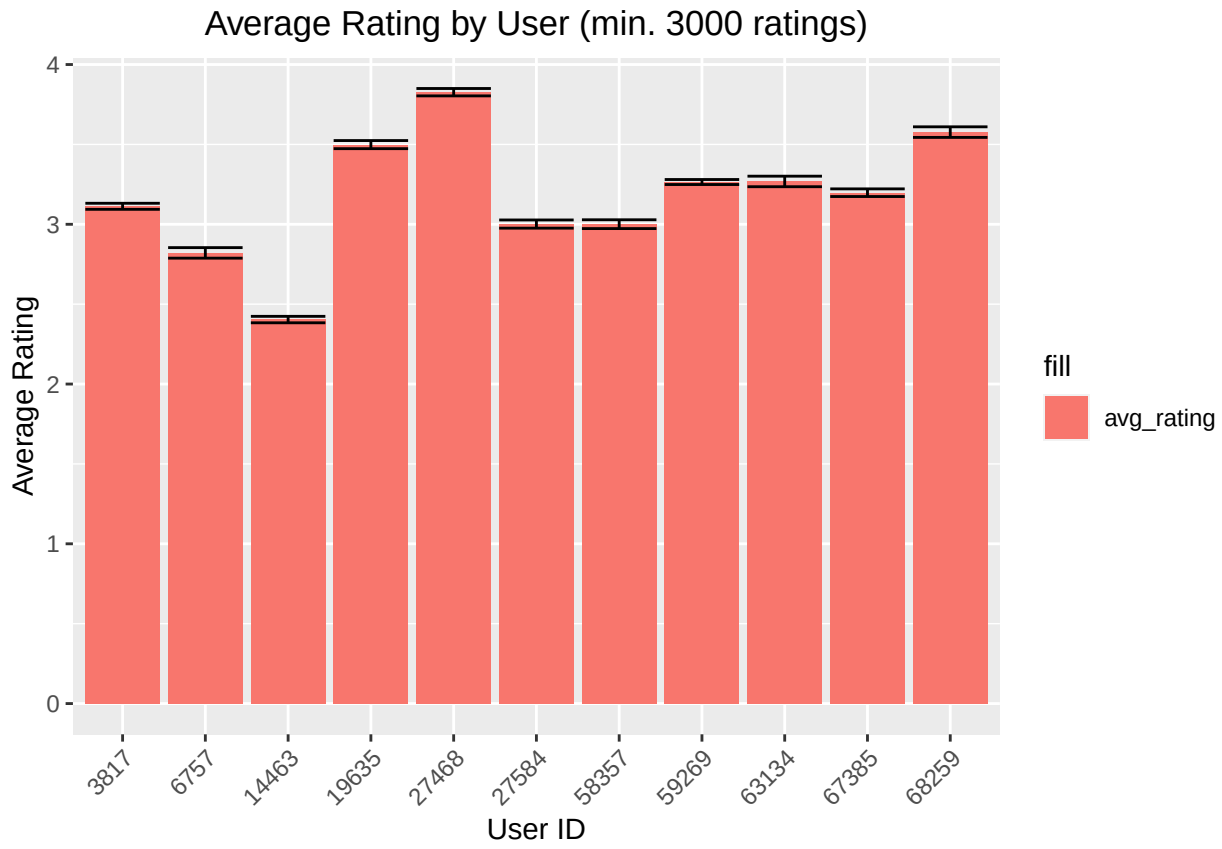
The plot below shows the overall distribution of ratings. It appears that the most popular ratings are 3 and 4 out of 5. Another interesting point is that ratings ending in .5 are not used as frequently as whole number ratings.



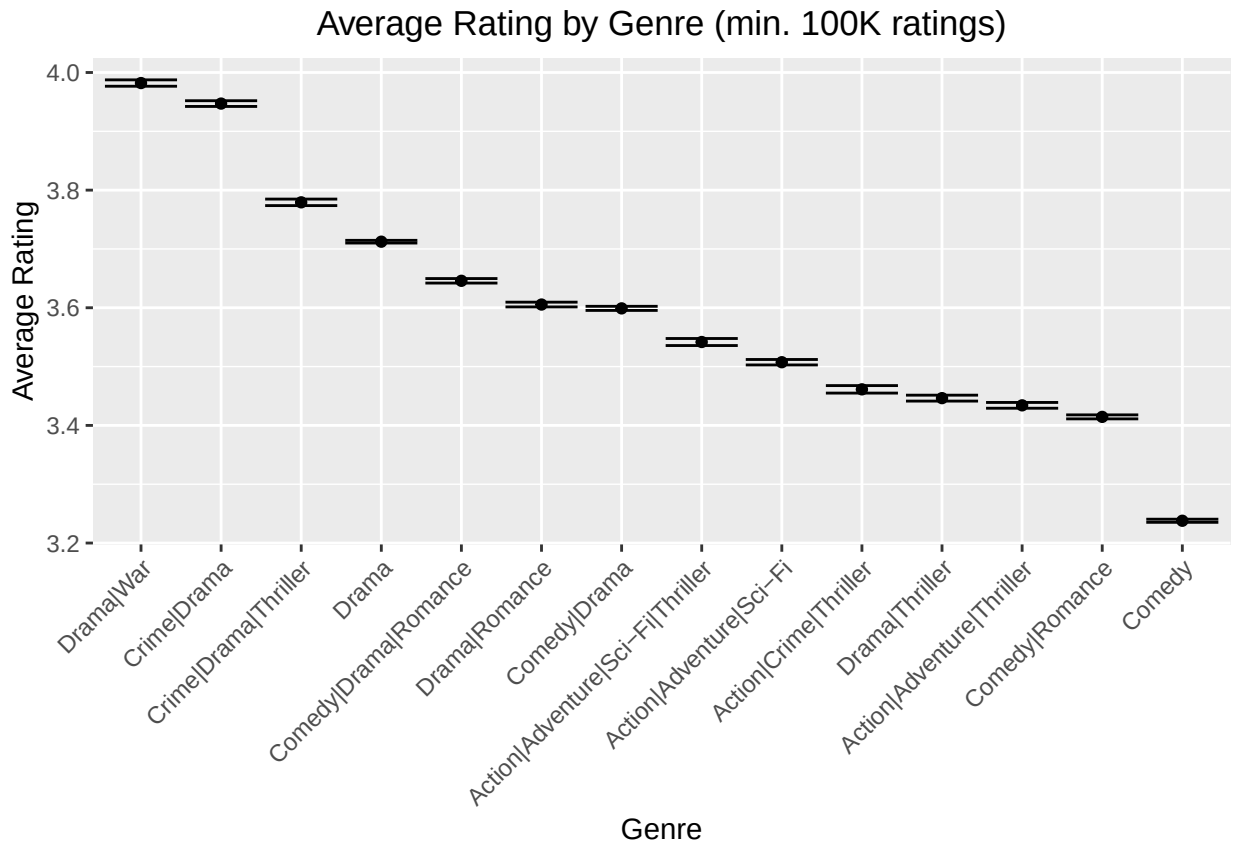
Now, we can examine the average rating for movies with many reviews. Among those movies with many ratings, there appears to be significant variation in the average rating. This helps show that we must take the ratings of different movies into account when predicting user ratings.



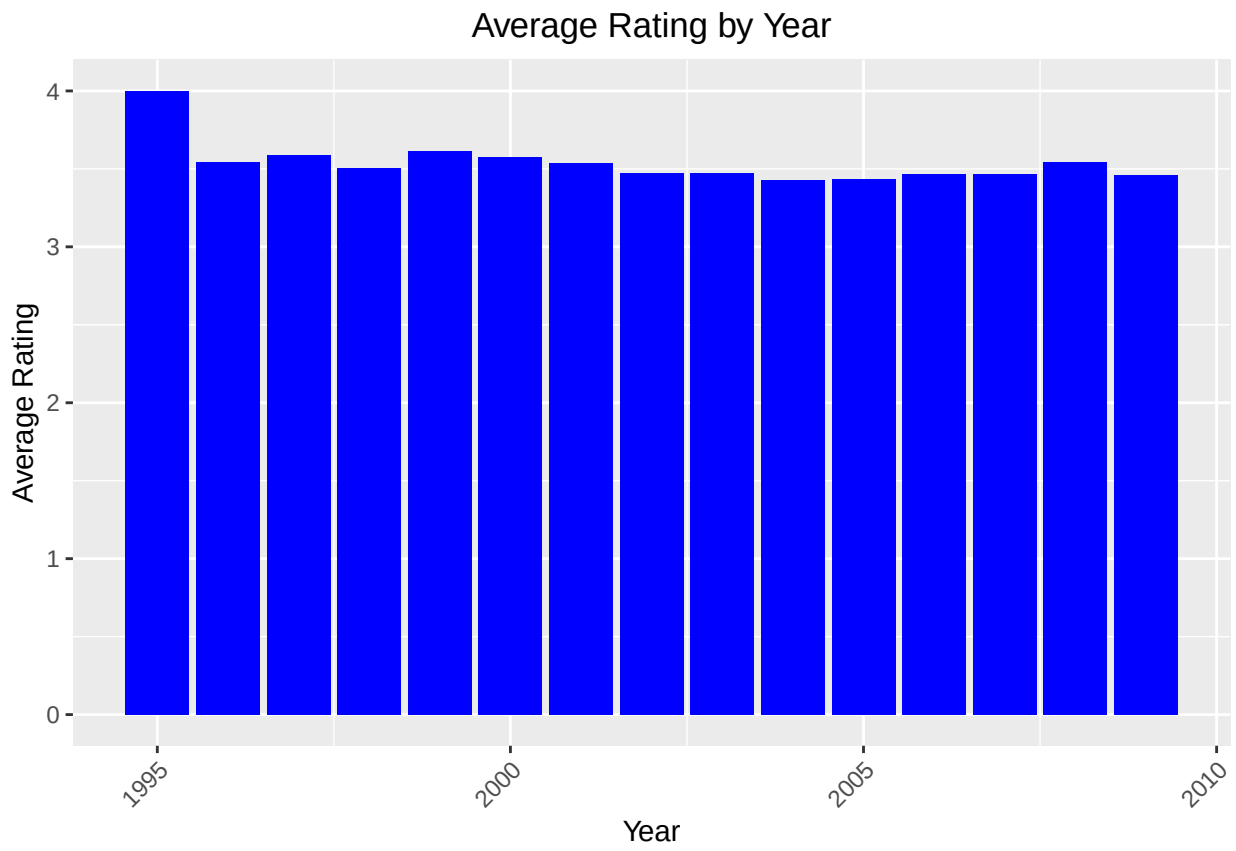
We can also examine the average rating for users with many reviews. Among those users with many ratings (at least 3000), there is a mix of harsh and lenient users. For instance, user 14463 tends to give much lower ratings, while user 27468 is much more lenient.



We can plot the ratings by each genre, but because there are too many different genre types, we only use the most common ones which have at least 100,000 ratings. The error bar plot below clearly shows that certain genres are more well-received than others. Clearly, comedy films receive lower ratings on average than drama.



There is also some bias based on which year the movie was rated From the graph below, it is evident that earlier reviews tend to receive higher ratings than recent ones.



Prediction Methods

For building our models, we first create a train and test set from the edx dataset. We follow the same steps as when splitting the MovieLens dataset into the edx and validation sets.

Our first attempt at predicting user ratings is to predict every rating to equal the average of all user ratings. In order to determine the effectiveness of the model, we compute the root mean squared error (RMSE), which is the standard deviation of the prediction errors. For all models, we evaluate the RMSE with respect to the test_edx set, and once the best model has been selected, we will evaluate on the validation set.

```
#Simple model based on overall average of all ratings
ovrl_avg = mean(train_edx$rating)

#Predict all movies to have the average rating
simple_RMSE = RMSE(test_edx$rating, ovrl_avg)
simple_RMSE
```

```
## [1] 1.059904
```

The RMSE for this model on the test set is 1.059904. This is our baseline RMSE, and any subsequent models should have a better (lower) RMSE than this model.

To improve the model, we can take the movie effects into account. We predict a user rating for a particular movie based on how other users have rated the same movie. For example, a really well-regarded classic film would generally have much higher ratings than the average. The model's predictions would accommodate the average ratings of each movie instead of just the overall average rating.

```
#Model based on movie effects
movie_avg = train_edx %>%
  group_by(movieId) %>%
  summarize(movie_i = mean(rating - ovrl_avg))

#Predict based on the average rating for each movie
movie_based_pred = test_edx %>%
  left_join(movie_avg, by = 'movieId') %>%
  mutate(pred = ovrl_avg + movie_i) %>%
  pull(pred)

movie_RMSE = RMSE(test_edx$rating, movie_based_pred)
movie_RMSE
```

```
## [1] 0.9437429
```

To further improve the model, we can add user effects to the movie effects. For example, a harsh critic will tend to give lower ratings, so we can predict his ratings to be lower than the average. Furthermore, we include movie effects as well to prevent the prediction from being too low for a generally well-received movie.

```
#Model based on movie and user effects
um_avg = train_edx %>%
  left_join(movie_avg, by = "movieId") %>%
  group_by(userId) %>%
  summarize(user_i = mean(rating - ovrl_avg - movie_i))

#Predict based on the average rating for each user and movie
um_pred = test_edx %>%
  left_join(movie_avg, by = 'movieId') %>%
  left_join(um_avg, by = 'userId') %>%
  mutate(pred_um = ovrl_avg + movie_i + user_i) %>%
  pull(pred_um)

um_RMSE = RMSE(test_edx$rating, um_pred)
um_RMSE
```

```
## [1] 0.8659319
```

One further adjustment we can make in the above model is to adjust rating predictions that are not within the acceptable range of 0.5-5. For example, there are over 4000 ratings that are predicted to be above 5. Instead, we can change these predictions to equal 5, since we know that ratings above 5 are not possible. Similarly, we can adjust predictions less than 0.5 to equal 0.5.

```
sum(um_pred > 5) #4311 ratings predicted above 5.0
```

```
## [1] 4311
```

```
sum(um_pred < 0.5) #229 ratings predicted below 0.5
```

```
## [1] 229
```

```
#Predict based on the average rating for each user and movie with adjustment for unacceptable ratings.
um_adj_pred = test_edx %>%
  left_join(movie_avg, by = 'movieId') %>%
  left_join(um_avg, by = 'userId') %>%
  mutate(pred_um = ovrl_avg + movie_i + user_i) %>%
  mutate(pred_um_adj = ifelse(pred_um > 5, 5.0,
                       ifelse(pred_um < 0.5, 0.5, pred_um))) %>%
  pull(pred_um_adj)

um_rmse_adj = RMSE(test_edx$rating, um_adj_pred)
um_rmse_adj
```

```
## [1] 0.865716
```

There is a minor improvement in the RMSE after this adjustment. Now, we can also try accounting for genre effects with the adjustment to ratings outside the acceptable range.

```
#Model based on movie, user, and genre effects
umg_avg = train_edx %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(um_avg, by = "userId") %>%
  group_by(genres) %>%
  summarize(genre_i = mean(rating - ovrl_avg - movie_i - user_i))

#Predict based on the average rating for each user, movie, and genre with adjustment
umg_pred_adj = test_edx %>%
  left_join(movie_avg, by = 'movieId') %>%
  left_join(um_avg, by = 'userId') %>%
  left_join(umg_avg, by = 'genres') %>%
  mutate(pred_umg = ovrl_avg + movie_i + user_i + genre_i) %>%
  mutate(pred_umg_adj = ifelse(pred_umg > 5, 5.0,
                        ifelse(pred_umg < 0.5, 0.5, pred_umg))) %>%
  pull(pred_umg_adj)

umg_RMSE_adj = RMSE(test_edx$rating, umg_pred_adj)
umg_RMSE_adj
```

```
## [1] 0.8653702
```

Now, let's try factoring in the effect of year into our model and adjust for ratings outside the acceptable range. To do this, we have to convert the timestamp to a date object using the as.Date.POSIXct function. Then we extract the year from this object.

```
#Model based on movie, user, genre, and year effects
umgy_avg = train_edx %>%
  left_join(movie_avg, by = "movieId") %>%
  left_join(um_avg, by = "userId") %>%
  left_join(umg_avg, by = "genres") %>%
  mutate(year = year(as.Date.POSIXct(timestamp))) %>%
  group_by(year) %>%
  summarize(year_i = mean(rating - ovrl_avg - movie_i - user_i -                        genre_i))
```

```r
#Predict based on the average rating for each user, movie, genre, and year
umgy_pred = test_edx %>%
  mutate(year = year(as.Date.POSIXct(timestamp))) %>%
  left_join(movie_avg, by = 'movieId') %>%
  left_join(um_avg, by = 'userId') %>%
  left_join(umg_avg, by = 'genres') %>%
  left_join(umgy_avg, by = 'year') %>%
  mutate(pred_umgy = ovrl_avg + movie_i + user_i + genre_i +                              year_i) %>%
  mutate(pred_umgy_adj = ifelse(pred_umgy > 5, 5.0,
                         ifelse(pred_umgy < 0.5, 0.5, pred_umgy))) %>%
  pull(pred_umgy_adj)

umgy_RMSE_adj = RMSE(test_edx$rating, umgy_pred)
umgy_RMSE_adj
```

```
## [1] 0.8653532
```

The RMSE for the model taking into account movie, user, genre, and year effects is **0.8653532**. It appears that by adding genre and year effects, we achieved slight improvements in RMSE. In the next section, we use regularization to try and get more improvements in RMSE.

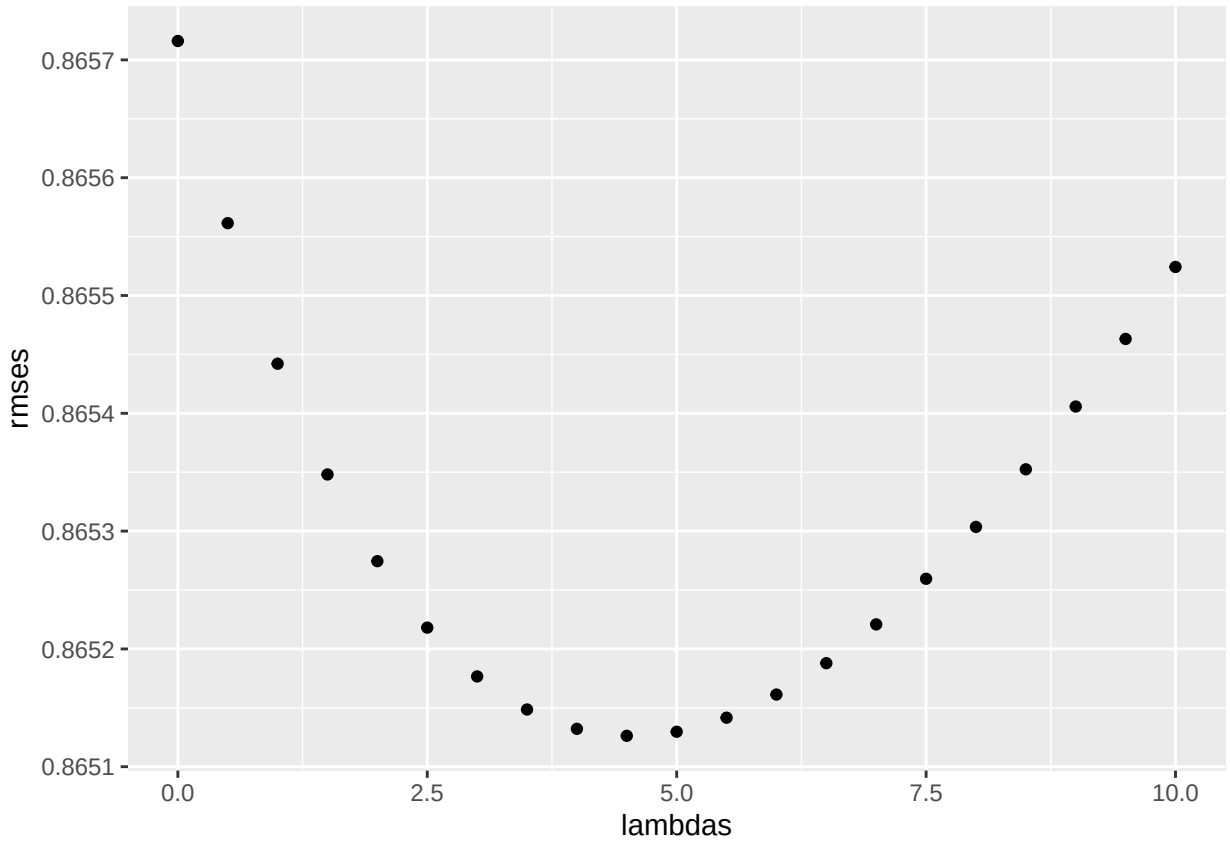
Regularization Methods

Until this point, the models we have constructed do not account for the sample size effect on our predictions. For example, a movie with very few ratings might be skewing the predictions in an unfavorable direction. With regularization, we can penalize these estimates and prevent them from having large impacts on our predictions.

We can start by regularizing the model which accounts for movie and user effects. We use a range of lambda values as the regularization parameter, and select the value which gives the lowest RMSE. We also generate a plot to visually show the lambda values against the RMSE values.

```r
lambdas <- seq(0, 10, 0.5)

rmses <- sapply(lambdas, function(l){

  mu <- mean(train_edx$rating)

  b_i <- train_edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  b_u <- train_edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  predicted_ratings <-
    test_edx %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred_um_reg = mu + b_i + b_u) %>%
    mutate(pred_um_reg_adj = ifelse(pred_um_reg > 5, 5.0,
```

```
                            ifelse(pred_um_reg < 0.5, 0.5,
                                   pred_um_reg))) %>%
    pull(pred_um_reg_adj)

  return(RMSE(predicted_ratings, test_edx$rating))
})

qplot(lambdas, rmses)
```




```
min_lambda_um_reg = lambdas[which.min(rmses)]
min_lambda_um_reg #lambda = 4.5 provides lowest RMSE
```

```
## [1] 4.5
```

```
min_rmse_um_reg = min(rmses)
min_rmse_um_reg
```

```
## [1] 0.8651262
```

For this regularized model accounting for movie and user effects, we see that lambda = 4.5 provides the lowest RMSE of **0.8651262**.

We attempt to regularize the model consisting of movie, user, genre, and year effects. We follow a similar technique as the previous regularized movie and user effect model, but just add in additional terms for genre and year.

```
lambdas <- seq(0, 10, 0.5)

rmses_umgy <- sapply(lambdas, function(l){

  mu <- mean(train_edx$rating) #overall average rating

  #effect of movie i
  b_i <- train_edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  #effect of user u
  b_u <- train_edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  #effect of genre g
  b_g <- train_edx %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))

  #effect of year y
  b_y <- train_edx %>%
    mutate(year = year(as.Date.POSIXct(timestamp))) %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - b_i - b_u - b_g - mu)/(n()+l))

  #predict the ratings for the test_edx set
  predicted_ratings <- test_edx %>%
    mutate(year = year(as.Date.POSIXct(timestamp))) %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_y, by = "year") %>%
    mutate(pred_umgy_reg = mu + b_i + b_u + b_g + b_y) %>%
    mutate(pred_umgy_reg_adj = ifelse(pred_umgy_reg > 5, 5.0,
                              ifelse(pred_umgy_reg < 0.5, 0.5,
                                     pred_umgy_reg))) %>%
    pull(pred_umgy_reg_adj)

  return(RMSE(predicted_ratings, test_edx$rating))
})

min_lambda_umgy_reg = lambdas[which.min(rmses_umgy)]
min_lambda_umgy_reg #lambda = 4.5 provides lowest RMSE
```

```
## [1] 4.5
```

```
min_rmse_umgy_reg = min(rmses_umgy)
min_rmse_umgy_reg #0.8647711
```

```
## [1] 0.8647711
```

A summary of the RMSE values for each model we have tried is shown below. The models were build using the train_edx set and tested on the test_edx set. For our final step, we will make our predictions on the validation set using the edx set.

Model	RMSE on test_edx
Average	1.059904
Movie Effect	0.9437429
Movie + User Effect	0.8659319
Movie + User Effect (adj.)	0.865716
Movie + User + Genre Effect (adj.)	0.8653702
Movie + User + Genre + Year Effect (adj.)	0.8653532
Regularized Movie + User (adj.)	0.8651262
Regularized Movie + User + Genre + Year Effect (adj.)	0.8647711

Results

After all the inital modeling efforts, we decided upon the best model to use, and now measure the performance of this model on the validation (test) set. The final model uses regularization and takes into account movie, user, genre, and year effects. It also adjusts any ratings outside the acceptable range.

The final model equation is as follows: $Y_{u,i} = \mu + b_i + b_u + \sum_{j=1}^{J} x_{u,i}^{j}\beta_j + \sum_{k=1}^{K} x_{u,i}^{k}\beta_k + adj + \varepsilon_{u,i}$

where

μ = average of all ratings

b_i = effect of movie i

b_u = effect of user u

$x_{u,i}^{j} = 1$ if $y_{u,i}$ (year for user u's ratings of movie i) is year j

β_j = effect of year j for $j = 1, ..., J$

$x_{u,i}^{k} = 1$ if $g_{u,i}$ (genre for user u's ratings of movie i) is genre k

β_k = effect of genre k for $k = 1, ..., K$

adj = the adjustment term to keep the predicted rating within 0.5 to 5.

$\varepsilon_{u,i}$ = independent errors sampled from the same distribution centered at 0

This model is regularized with $\lambda = 4.5$ chosen as the regularization parameter.

```
#Use the regularized user, movie, genre, and year model
#Build the model using edx set. Test the model on the validation set.
lambda <- 4.5 #Selected from model in previous section

rmses_umgy_edx_fn <- function(l){
  # This function constructs the regularized model on the edx set
  # given input lambda (l) and calculates RMSE on the validation set
```

```
  mu <- mean(edx$rating) #overall average rating

  #effect of movie i
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))

  #effect of user u
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  #effect of genre g
  b_g <- train_edx %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - b_i - b_u - mu)/(n()+l))

  #effect of year y
  b_y <- edx %>%
    mutate(year = year(as.Date.POSIXct(timestamp))) %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    group_by(year) %>%
    summarize(b_y = sum(rating - b_i - b_u - b_g - mu)/(n()+l))

  #predict the ratings for the validation set
  predicted_ratings <- validation %>%
    mutate(year = year(as.Date.POSIXct(timestamp))) %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_y, by = "year") %>%
    mutate(pred_umgy_reg = mu + b_i + b_u + b_g + b_y) %>%
    mutate(pred_umgy_reg_adj = ifelse(pred_umgy_reg > 5, 5.0,
                                      ifelse(pred_umgy_reg < 0.5, 0.5,
                                             pred_umgy_reg))) %>%
    pull(pred_umgy_reg_adj)

  ratings_and_RMSE_list = list(head(predicted_ratings, 50),
                               RMSE(predicted_ratings, validation$rating))
  return(ratings_and_RMSE_list)
}

#Return the ratings and RMSE on the validation set using lambda = 4.5
rmses_umgy_edx_fn(lambda) #0.8642996
```

```
## [[1]]
##  [1] 4.307110 5.000000 4.410230 3.360620 4.264484 2.744594 3.927166 4.165007
##  [9] 4.265257 3.299972 3.642649 3.610485 3.712534 4.163593 3.521791 4.255255
```

```
## [17] 3.786224 3.434416 4.271013 3.904882 3.843042 4.121834 4.201592 4.290408
## [25] 4.513662 3.393714 3.124534 4.085048 4.360663 4.356702 4.188423 4.006070
## [33] 4.119466 3.808060 3.898502 3.830669 4.037481 3.326925 3.463798 4.270235
## [41] 4.055240 3.452303 3.071099 3.377085 3.156209 3.688146 3.898913 4.309324
## [49] 3.181228 3.837882
##
## [[2]]
## [1] 0.8642996
```

The results show some of the ratings predicted by users. Our final model provides an RMSE of **0.8642996** on the validation set, which is our best result.

Conclusion

Through careful analysis and modeling efforts, we have provided a way to predict user ratings for movies. The driving factors for our prediction are the movie and user effects. The movie effect allows us to predict the ratings based on the overall ratings of a movie, while the user effect allows us to predict the ratings based on how harshly a user tends to review. Additional variables for genre and year were included based on exploratory analysis which there are some patterns to exploit from these variables. Our adjustment to keep all predicted ratings within the scale of 0.5 to 5 is a sort of sanity check which keeps predictions from being too far off the mark. Lastly, regularization was applied to reduce impacts of variables with small sample size from skewing the predictions.

References

https://rafalab.github.io/dsbook/large-datasets.html#recommendation-systems Some of the modeling code was based off of code provided in the course textbook under the Recommendation Systems and Regularization Sections. All code used for this project can be viewed in the R script file in this directory.