Code

MovieLens Project

Melody C. Brown, MHSA, MPH, CSPO

March 2022

Introduction

In 2006, the Netflix Movie challenge offered a prize of a million dollars to anyone who could improve their recommendation system algorithm by 10%. The goal of this project is to replicate a similar challenge using a subset of the publicly available data set generated by the GroupLens research lab to improve a theoretical model. The challenge for this project was to create a model that will achieve a Root Mean Square Error (RMSE) as close to or lower than 0.86490 using only the and data included in the prescribed data set.

In approaching this challenge, it's important to first consider the available information, which includes user IDs, ratings, titles, time stamps, and genres. Not all users rated every movie and many movies were unrated by users, which leads to the assumption that the movies rated were generally based on popularity (i.e., the more popular the movie, the more ratings it would most likely get) and/or the individual users' general preferences of movies and genre types (i.e., users would be more likely to watch and rate movies and genres that they like). The approach for this project was to use a combination of movie (e.g. average ratings per movie title), user (e.g., average ratings per user), and genre effects (e.g., based on average user ratings by genre) to predict movies that could be recommended to users based off their previous ratings.

To begin this challenge, we will first explore the data and discuss insights gained towards selecting a modeling approach. The data will then be manipulated to set up the model to be applied to the data set for validation, and the performance results will be discussed. Final conclusions and suggestions for future work will also be included.

Methods

This project used the MovieLens 10M data set, that is described here (https://grouplens.org/datasets/movielens/10m/) and was downloaded from here (http://files.grouplens.org/datasets/movielens/ml-10m.zip). The data file was first processed according to the project instructions to be reformatted and to create the validation set to be used to test the final model.

Hide

```
## Read in data file
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

## Rename and reformat Columns
ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))
movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")

## Convert to a data frame and join new columns
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(movieId),
                                           title = as.character(title),
                                           genres = as.character(genres))
movielens <- left_join(ratings, movies, by = "movieId")

## Set Validation set as 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.5 or earlier, use `set.seed(1)`
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

Hide

```r
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

#Confirm userId and movieId in validation set are also in edx set
validation <- temp %>%
      semi_join(edx, by = "movieId") %>%
      semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

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

Edx Sample Data Set Exploration

The data set is comprised of 9,000,055 observations with a total of 69, 878 distinct users and 10,677 unique movie titles across the following six variables:
- **userID:** unique, numeric IDs attached to individual users to indicate each rating by title
- **movieID:** unique, numeric IDs attached to individual movie titles
- **rating:** numeric ratings in 0.5 intervals, with a range of 0.5 to 5.0 and average rating of 3.512
- **timestamp:** numeric date and time rating was given
- **title:** movie titles and dates of release of each
- **genres:** genres associated with each title (can be multiple per title); 20 possible genres available

Hide

```r
# Glimpse of data set
glimpse(edx)
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, …
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420, …
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, …
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83898…
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", "S…
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|Sci…
```

Hide

```r
# Range of ratings
descriptives <-edx %>%
  summarize(n_users = n_distinct(userId),
           n_movies= n_distinct(movieId),
           mean_rating = mean(rating),
           min_rating = min(rating),
           max_rating = max(rating))

knitr::kable(descriptives, caption="Descriptives Summary Table")
```

Descriptives Summary Table

n_users	n_movies	mean_rating	min_rating	max_rating
69878	10677	3.513	0.5	5

Frequency of Ratings by Title

The top rated movies that received over 30,000 ratings were Pulp Fiction, Forrest Gump, and Silence of the Lambs.

Hide

```
#Calculate frequency of ratings by title
topratingsdf <-edx %>% group_by(title) %>%
  summarize(count = n()) %>%
  top_n(25) %>%
  arrange(desc(count))

#Create table of top 25 rated movie titles by count
trs <-head(topratingsdf, 25)
knitr::kable(trs, caption="Top 25 Rated Movie Titles by Count")
```

Top 25 Rated Movie Titles by Count

title	count
Pulp Fiction (1994)	31362
Forrest Gump (1994)	31079
Silence of the Lambs, The (1991)	30382
Jurassic Park (1993)	29360
Shawshank Redemption, The (1994)	28015
Braveheart (1995)	26212
Fugitive, The (1993)	25998
Terminator 2: Judgment Day (1991)	25984
Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)	25672
Apollo 13 (1995)	24284
Batman (1989)	24277
Toy Story (1995)	23790
Independence Day (a.k.a. ID4) (1996)	23449
Dances with Wolves (1990)	23367
Schindler’s List (1993)	23193
True Lies (1994)	22823
Star Wars: Episode VI - Return of the Jedi (1983)	22584
12 Monkeys (Twelve Monkeys) (1995)	21891
Usual Suspects, The (1995)	21648
Fargo (1996)	21395
Speed (1994)	21361
Aladdin (1992)	21173
Matrix, The (1999)	20908
Star Wars: Episode V - The Empire Strikes Back (1980)	20729
Seven (a.k.a. Se7en) (1995)	20311

To further explore the frequency of movies that were rated, we should also look at how often users rated multiple movies and at what rate. To do so, the ratings frequency will be divided into levels with a distance of 5,000 ratings at each level between under 5,000 ratings and over 30,000 to provide a meaningful visualization. The resulting graph shows that over 96% of movie titles in the data set received less than 5,000 rating scores. This seems to indicate that a large number of movies in the data set received very few scores that were most likely very variable between each user based on their preferences, while the more frequently rated movies should normalize.

Hide

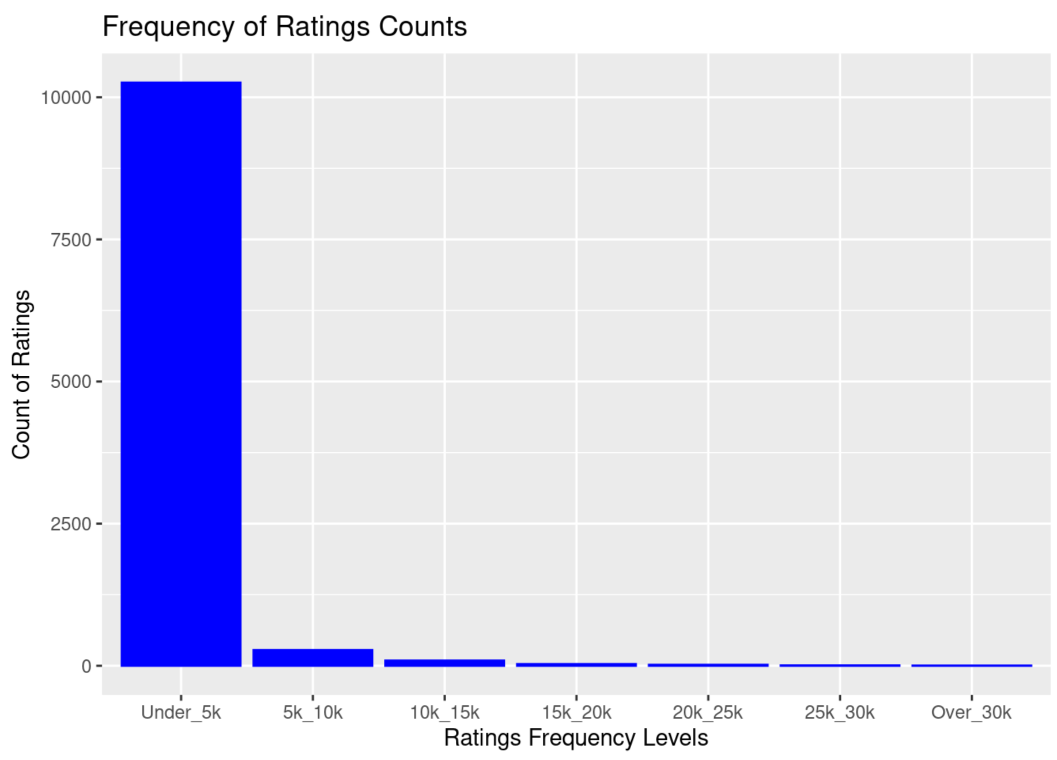
```
##Convert frequencies to level and then plot to see distribution ratings frequency in a visually meaningful way
topratingsdf <- edx %>%
  group_by(title) %>%
  summarize(count = n()) %>%
  arrange (desc(count)) %>%
  as.data.frame()

#Convert count to numeric
topratingsdf$count <- as.numeric(unlist(topratingsdf$count))

#Add Ratings Frequency Levels
ratingsfreqdf <-topratingsdf %>% mutate(RatingsFreq = cut(count, breaks=c(0, 5000, 10000, 15000, 20000, 25000, 30000, 40000
), labels= c("Under_5k", "5k_10k", "10k_15k", "15k_20k", "20k_25k", "25k_30k", "Over_30k")))

#Percentage of Ratings by Level
ratingsfreqtbl <- prop.table(table(ratingsfreqdf$RatingsFreq))* 100

knitr::kable(ratingsfreqtbl, caption="Percentage of Ratings by Level")
```

Percentage of Ratings by Level

Var1	Freq
Under_5k	96.0753
5k_10k	2.5852
10k_15k	0.8430
15k_20k	0.2623
20k_25k	0.1499
25k_30k	0.0562
Over_30k	0.0281

Hide

```
#Plot frequency of ratings
ggplot(ratingsfreqdf, aes(RatingsFreq)) +
  labs(title= "Frequency of Ratings Counts", x= "Ratings Frequency Levels", y="Count of Ratings")+
  geom_bar(color="blue", fill="blue")
```



Hide

```
rm(descriptives, topratingsdf, ratingsfreqtbl, trs)
```

Genres

Drama and Comedy were the most common genre type, followed by Action, Thriller, and Adventure; the least common types included Western, Film Noir, Documentary, and IMAX. The assumption can therefore be made that less common genres are more likely to be influenced by user preferences and more variable while more common genres should be more normalized in terms of ratings. Average user ratings by genre could therefore be an important parameter to use within the model, and will be explored further.

Hide

```
## Remove timestamp and title columns from edx and validation sets to optimize processing time
edx <- within(edx, rm(timestamp, title))
validation <- within(validation, rm(timestamp, title))

#Split genres into distinct rows for analysis and ability to calculate information based on genres
edx <- edx %>% separate_rows(genres, sep = "\\|")
validation <- validation %>% separate_rows(genres, sep = "\\|")

# Unique genres
uniquegenresdf <- edx %>%
  group_by(genres) %>%
  summarize(count=n()) %>%
  arrange(desc(count))

knitr::kable(uniquegenresdf, caption="Unique Genres by Count")
```

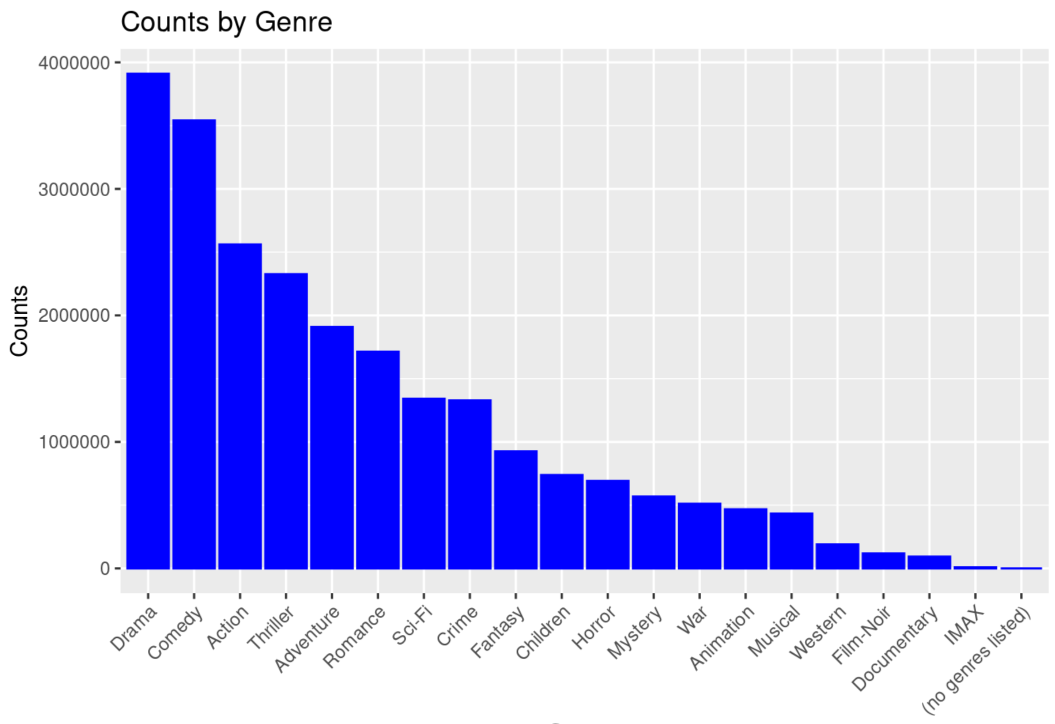
Unique Genres by Count

genres	count
Drama	3910127
Comedy	3540930
Action	2560545
Thriller	2325899

genres	count
Adventure	1908892
Romance	1712100
Sci-Fi	1341183
Crime	1327715
Fantasy	925637
Children	737994
Horror	691485
Mystery	568332
War	511147
Animation	467168
Musical	433080
Western	189394
Film-Noir	118541
Documentary	93066
IMAX	8181
(no genres listed)	7

Hide

```
#Plot of genres by count
ggplot(uniquegenresdf, aes(reorder(x =genres, -count), y = count)) +
  labs(title = "Counts by Genre", x = "Genres", y = "Counts") +
  scale_y_continuous(labels = function(x) format(x, scientific = FALSE))+
  geom_bar(stat = "identity", color = 'blue', fill= 'blue') +
  theme(axis.text.x = element_text(angle=45, hjust=1.0))
```



Hide

```
rm(uniquegenresdf)
```

Procedure

The general approach to this project will be to use a multivariate regression model to predict a rating for movie *y~g,i* by creating a model that includes , the average rating; three bias parameters of_b~i_, a movie effect (defined by the difference in the average rating versus average rating by movie title), _b~u, a user effect (defined by the difference between the user's ratings versus the average ratings), and _b$_g$, a genre effect (defined by the difference in average user genre ratings versus average rating). An error parameter is included to account for the variations based on some movies and genres receiving many more ratings than others. The proposed base model can be defined as follows:

$$y_{g,i} = \mu + b_i + b_u + b_g + \epsilon_{g,i}$$

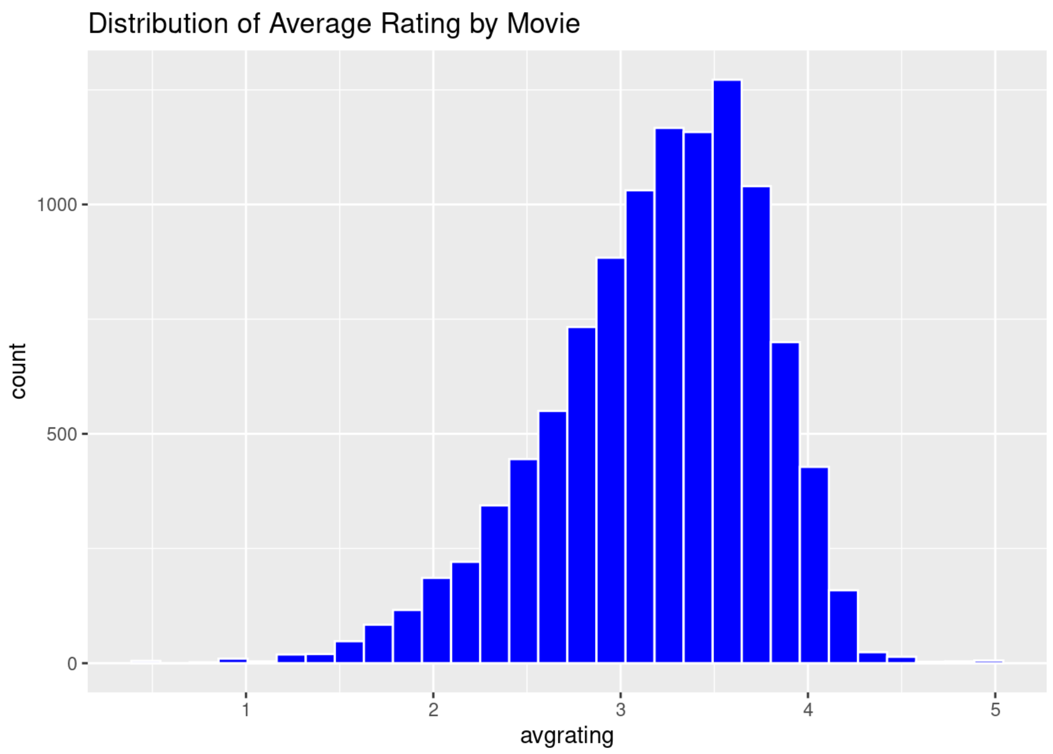
Data Preparation and Final Visualizations

To prepare for analysis and model building, the data set was first manipulated to remove time stamp and title columns since they are not used in this analysis and to optimize processing abilities (see R code provided in the Methods section under the Genres subsection). Then, the genre column was separated from having multiple genres listed in one column so that the genres could be analyzed.

Next, the average ratings by movie are calculated and presented visually along with the distribution of ratings by genre. When looking at the distribution of average ratings by movie, we can see a left-skewed distribution.

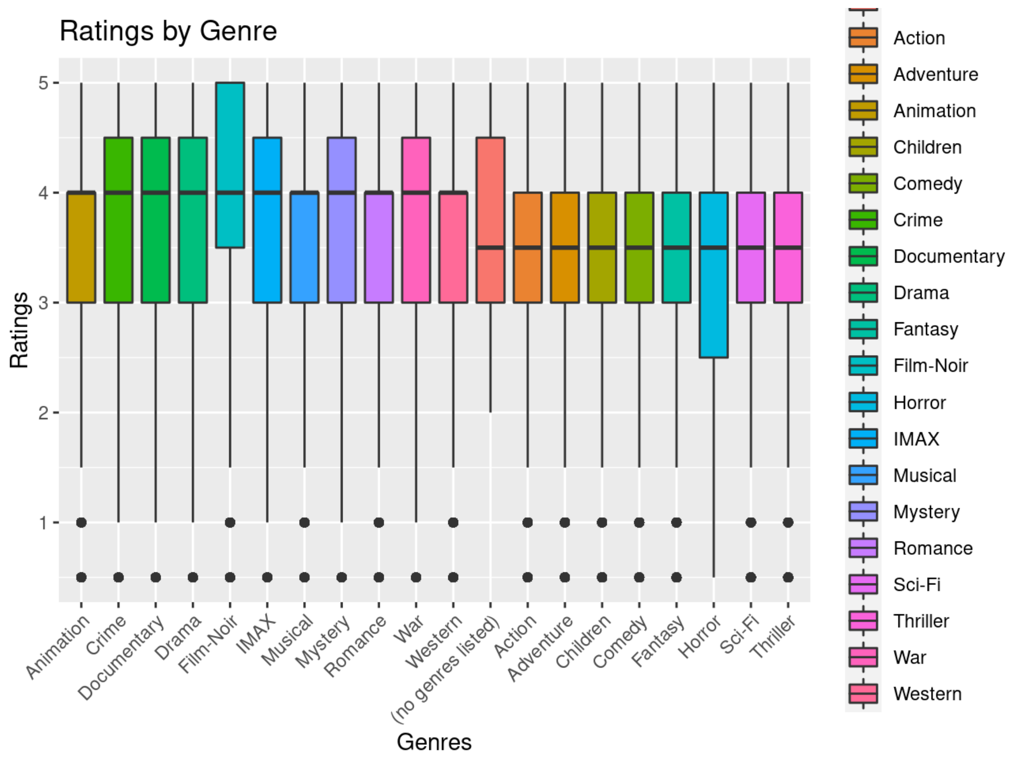
Hide

```
#Calculate average rating by movie ID
avgrating <- edx %>%
  group_by(movieId) %>%
  summarize(avgrating = mean(rating))

#Visualize average rating by movie distribution
ggplot(avgrating, aes(x=avgrating))+
  labs(title="Distribution of Average Rating by Movie") +
  geom_histogram(col="white", fill="blue")
```



Hide

```
#Boxplot of Ratings by genres
grplot <-edx %>%
  ggplot(aes(reorder(x = genres, -rating, median), y = rating, fill = genres)) +
  labs(title = "Ratings by Genre", x = "Genres", y = "Ratings") +
  geom_boxplot()+
  theme(axis.text.x = element_text(angle=45, hjust=1.0))
grplot
```



Hide

```
rm(grplot, avgrating)
```

Exploratory Model Analysis

Partition Edx Data Set

To test the model, we will first create a test set made up of 10% of the edx data set to explore the accuracy of the proposed model. We will also remove the time stamp and title columns to help minimize the size of the data file for better processing.

Hide

```
## Set test set as 10% of the edx data set
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

Hide

```
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
edx_temp <- edx[test_index,]

# Confirm that userId and movieId are in the train and test sets
test_set <- edx_temp %>%
      semi_join(train_set, by = "movieId") %>%
      semi_join(train_set, by = "userId")

# Add rows removed from the test set into train set
edx_removed <- anti_join(edx_temp, test_set)
train_set <- rbind(train_set, edx_removed)

rm(edx_temp, test_index, edx_removed)
```

Building the Model

To begin building the model, we will first need to define the RMSE. The RMSE will be calculated to determine the error made in predicting movie ratings based on the model created, with a goal of achieving as close to or lower than 0.86490. The RMSE is defined as the following:

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

Hide

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

To establish a baseline, we will first build a simple model to predict the ratings that assumes the same rating for all movies and users and any variations accounted for by chance, with independent errors centered at 0:

$$Y_{g,i} = \mu + \epsilon_{g,i}$$

Baseline Model

This model provides a baseline to compare how well future ratings could be predicted if we only used the mean to train the model.

Hide

```
## Calculate mu, the average of all ratings
mu <- mean(train_set$rating)

##Create base model
basemodel <- RMSE(test_set$rating, mu)

#Create test results table
RMSE_Results <- tibble(Method= "Base Model", RMSE = basemodel)

knitr::kable(RMSE_Results, caption ="Model Results Table")
```

Model Results Table

Method	RMSE
Base Model	1.052

Movie Effects

The first penalty term that we will try is the movie effect, where the average rating per movie will be used to define the parameter. The model can be defined as:

$$y_{g,i} = \mu + b_i + \epsilon_{g,i}$$

This helps improve the RMSE over the baseline model to 0.94, but it’s not enough to reach the target.

Hide

```
## Calculate average user's rating by movie & calculating b_i
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating-mu))

##Check prediction against test set
prediction_bi <- mu + test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  pull(b_i)

#Calculate RMSE Results
MovieModel <- RMSE(prediction_bi, test_set$rating)

#Create test results table
RMSE_Results <- tibble(Method= c("Base Model", "Movie Model"), RMSE = c(basemodel, MovieModel))

knitr::kable(RMSE_Results, caption ="Model Results Table")
```

Model Results Table

Method	RMSE
Base Model	1.0517
Movie Model	0.9407

Movie + User Effects

The second parameter added is the user effect, where the users' average ratings will be used to define the parameter. This model can be defined as:

$$y_{g,i} = \mu + b_i + b_u + \epsilon_{g,i}$$

This also improves the RMSE to under 0.860, which is below the target.

Hide

```
## Calculate average user's rating by movie & calculating b_i
user_avgs <- train_set %>%
  left_join(movie_avgs, by="movieId")%>%
  group_by(userId) %>%
  summarize(b_u = mean(rating-mu -b_i))

##Check prediction against test set
prediction_bu <- test_set %>%
  left_join(movie_avgs, by ="movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  summarize(pred= mu+b_i +b_u) %>%
  pull(pred)

#Calculate RMSE Results
UserMovieModel <- RMSE(prediction_bu, test_set$rating)

#Create test results table
RMSE_Results <- tibble(Method= c("Base Model", "Movie Model", "Movie + User Model"), RMSE = c(basemodel, MovieModel, UserMo
vieModel))

knitr::kable(RMSE_Results, caption ="Model Results Table")
```

Model Results Table

Method	RMSE
Base Model	1.0517

Method	RMSE
Movie Model	0.9407
Movie + User Model	0.8568

Movie + User + Genre Effects

The third penalty effect that we will add will be the genre effect, where the users' average ratings per genre will be used to calculate the parameter. This model can be defined as:

$$y_{g,i} = \mu + b_i + b_u + b_g \epsilon_{g,i}$$

There were errors due to a possible inaccuracy of the join() function to properly join the data due to an insufficiently coded genre. This issue was unable to be resolved within the given time frame. This unfortunately did not improve the RMSE, and instead made it worse with an RMSE of 1.28.

Hide

```
## Calculate average user's rating by genre & calculating b_g
genre_avgs <- train_set %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(user_avgs, by="userId") %>%
  group_by(userId, genres) %>%
  summarize(b_g = mean(rating-mu-b_i-b_u))

##Check prediction against test set
prediction_bg <- test_set %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(user_avgs, by ="userId", "genre") %>%
  left_join(genre_avgs, by = "userId") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)

#Calculate RMSE Results
MovieGenreModel <- RMSE(prediction_bg, test_set$rating)
```

```
## Warning in true_ratings - predicted_ratings: longer object length is not a
## multiple of shorter object length
```

Hide

```
MovieGenreModel
```

```
## [1] 1.275
```

Hide

```
#Create test results table
RMSE_Results <- tibble(Method= c("Base Model", "Movie Model", "Movie + User Model", "Movie + User + Genre Model"), RMSE = c
(basemodel, MovieModel, UserMovieModel, MovieGenreModel))

knitr::kable(RMSE_Results, caption ="Model Results Table")
```

Model Results Table

Method	RMSE
Base Model	1.0517
Movie Model	0.9407
Movie + User Model	0.8568
Movie + User + Genre Model	1.2754

Final Model Results

Since the last penalty term did not improve the model, only the movie effect and user effect will be used in the final model since they produced the best results. Now, we will apply the model to confirm similar results on the validation data set.

Baseline Model

We first confirm the baseline model, which produces similarly expected results from the test models.

Hide

```
## Calculate mu, the average of all ratings
edx_mu <- mean(edx$rating)

##Create base model
Finalbasemodel <- RMSE(validation$rating, edx_mu)

#Create test results table
RMSE_Results_Final <- tibble(Method= "Base Model", RMSE = Finalbasemodel)

knitr::kable(RMSE_Results_Final, caption ="Final Model Results Table")
```

Final Model Results Table

Method	RMSE
Base Model	1.053

Movie Effects

Now we add the movie effect, which also produces similar scores that improve upon the base but do not reach our target RMSE.

Hide

```
## Calculate average user's rating by movie & calculating b_i
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating-edx_mu))

##Check prediction against validation set
prediction_bi <- edx_mu + validation %>%
  left_join(movie_avgs, by = "movieId") %>%
  pull(b_i)

#Calculate RMSE Results
FinalMovieModel <- RMSE(prediction_bi, validation$rating)

#Create test results table
RMSE_Results_Final <- tibble(Method= c("Base Model", "Movie Model"), RMSE = c(Finalbasemodel, FinalMovieModel))

knitr::kable(RMSE_Results_Final, digits = 4, caption ="Final Model Results Table")
```

Final Model Results Table

Method	RMSE
Base Model	1.0526
Movie Model	0.9411

Movie + User Effects

The second effect is added and produces a similarly effective model as in the test set, producing an RMSE of 0.8634, which is below the target.

Hide

```
## Calculate average user's rating by movie & calculating b_i
user_avgs <- edx %>%
  left_join(movie_avgs, by="movieId")%>%
  group_by(userId) %>%
  summarize(b_u = mean(rating-edx_mu -b_i))

##Check prediction against validation set
prediction_bu <- validation %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId")%>%
  mutate(pred= edx_mu + b_i + b_u) %>%
  pull(pred)

#Calculate RMSE Results
FinalUserMovieModel <- RMSE(prediction_bu, validation$rating)

#Create test results table
RMSE_Results_Final <- tibble(Method= c("Base Model", "Movie Model", "Movie + User Model"), RMSE = c(Finalbasemodel, FinalMo
vieModel, FinalUserMovieModel))

knitr::kable(RMSE_Results_Final, caption ="Final Model Results Table")
```

Final Model Results Table

Method	**RMSE**
Base Model	1.0526
Movie Model	0.9411
Movie + User Model	0.8634

Movie + User + Genre Effects

This effect was not used in the final model due to the worsened RMSE value presented in the test models. The code is included here for illustrative purposes and for future work to improve the code so that the join() function could be used properly.

Hide

```
###NOT CALCULATED AS FINAL MODEL DUE TO LACK OF IMPROVEMENT

## Calculate average user's rating by genre & calculating b_g
genre_avgs <- edx %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(user_avgs, by="userId") %>%
  group_by(userId, genres) %>%
  summarize(b_g = mean(rating-edx_mu-b_i-b_u))


##Check prediction against validation set
prediction_bg <- validation %>%
  left_join(movie_avgs, by="movieId") %>%
  left_join(user_avgs, by ="userId", "genre") %>%
  left_join(genre_avgs, by = "userId") %>%
  mutate(pred = edx_mu + b_i + b_u + b_g) %>%
  pull(pred)


#Calculate RMSE Results
FinalMovieGenreModel <- RMSE(prediction_bg, validation$rating)

#Create test results table
RMSE_Results_Final <- tibble(Method= c("Base Model", "Movie Model", "Movie + User Model", "Movie + User + Genre Model", RMS
E = c(Finalbasemodel, FinalMovieModel, FinalUserMovieModel, FinalMovieGenreModel)))

knitr::kable(RMSE_Results_Final, caption ="Final Model Results Table")
```

Conclusion

The combined model included penalty effects for average ratings of movies, users' average ratings by movie, and users' average ratings by genre. The use of all three penalty effects in the test model unfortunately did not result in an RMSE of less than 0.900, most likely due to potential collinearity effects or due to a mismatch of this approach to determining the true effects of differences in genres. The join() function also had difficulty working correctly in joining on genres in the last model, and so the resulting RMSE could be inaccurate if there was an incorrect join.

The test model that only included the movie and user effects did, however, achieve an RMSE of under 0.8649. The final model used for the validation was therefore the model that included just the movie and user penalty effects, leaving out the genre effects. This model was able to achieve the best RMSE out of the three attempted of 0.8634. In the future, other models could include using matrix factorization to better examine the genre effects more accurately and also include regularization to further improve the results.

Limitations

One of the major limitations faced during this analysis was the lack of knowledge/time to gain that knowledge on how to best address the join() function issues and the limitations it experienced when dealing with the genre factor. This was most likely due to a few-to-many issue with the genres not having a proper ID term to individually match each movie, user, genre combination. This should be explored further by anyone seeking to use a similar approach for this data set or similar data sets. Another limitation around the use of genre included the separation of genres into unique rows for each movie instead of examining the different genre combinations. The decision to only focus on genres individually was due to the potential complexities and extended processing time/power required when working with so many different combinations (over 700). This is another area that could be explored further in the future with different approaches that may lead to more accurate predictions based on genre.