HarvardX Capstone Project: Movie Rating Prediction Using Movielens Dataset

Goodarz Ghanavati

10/12/2020

Introduction

This report describes the method and the code developed for the HarvardX Movielens Capstone project. The project aim is to develop a movie recommendation system that predicts movie ratings using the information available in the 10M version of the Movielens dataset. The dataset, which is a subset of the Movielens dataset developed by the GroupLens research lab, includes information about users, movie ratings, movie genres, movie titles and release years.

The method described here builds upon the algorithm presented in the HarvardX Machine Learning course. Data exploration and visualization performed shows that there is a correlation between a movie's ratings and its age and genre. Therefore, in addition to the user and movie data, this method uses the genre and movie age as predictors. It also expands the use of Regularization to the genre and movie age predictors.

Method and Analysis

Create Dataset

The code below shows the method to create the Edx and Validation (the final hold-out test set) data sets from the Movielens dataset.

```
############################################################
# Create edx set, validation set (final hold-out test set)
############################################################

# Install required libraries
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(lubridate)) install.packages("lubridate", repos = "http://cran.us.r-project.org")
if(!require(stringr)) install.packages("stringr", repos = "http://cran.us.r-project.org")

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip
library(stringr)
library(dplyr)
library(caret)
library(lubridate)
library(stringr)
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)
```

```
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")

# # if using R 3.6 or earlier
# movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
#                                            title = as.character(title),
#                                            genres = as.character(genres))
# if using R 4.0 or later
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")

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

The Edx dataset was used for training the algorithm. For a final test of the algorithm, movie ratings in the validation set were predicted as if they were unknown.

In order to use the movie age as a predictor, I categorized movies based on their age. First, I extracted the movie release year from the title. I calculated the movie age by subtracting the release year from the current year (2020). Then, I categorized movies into five bins based on their age and added the movie age and age category to the Edx dataset.

```
# Extract the movie release year from the title
movie_year <- edx$title %>% str_extract("\\(\\d{4}\\)") %>%
  str_extract("(\\d{4})")
# Calculate the movie age
edx        <-  mutate(edx, movie_age = 2020-as.numeric(movie_year))

# Categorizing movies based on their age
edx$age_bins <- cut(edx$movie_age, breaks=c(0,20,40,60,80,105),
                    labels=c("1-20","21-40","41-60","61-80","81-105"))
```

The next step is to create the training and test sets from the Edx dataset. Using the semi_join function ensures that users and movies that do not appear in the training set are not included in the test set.

```r
# Create training and test sets
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]

# To make sure I don't include users and movies in the test set that do not
# appear in the training set, I removed these using the semi_join function
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

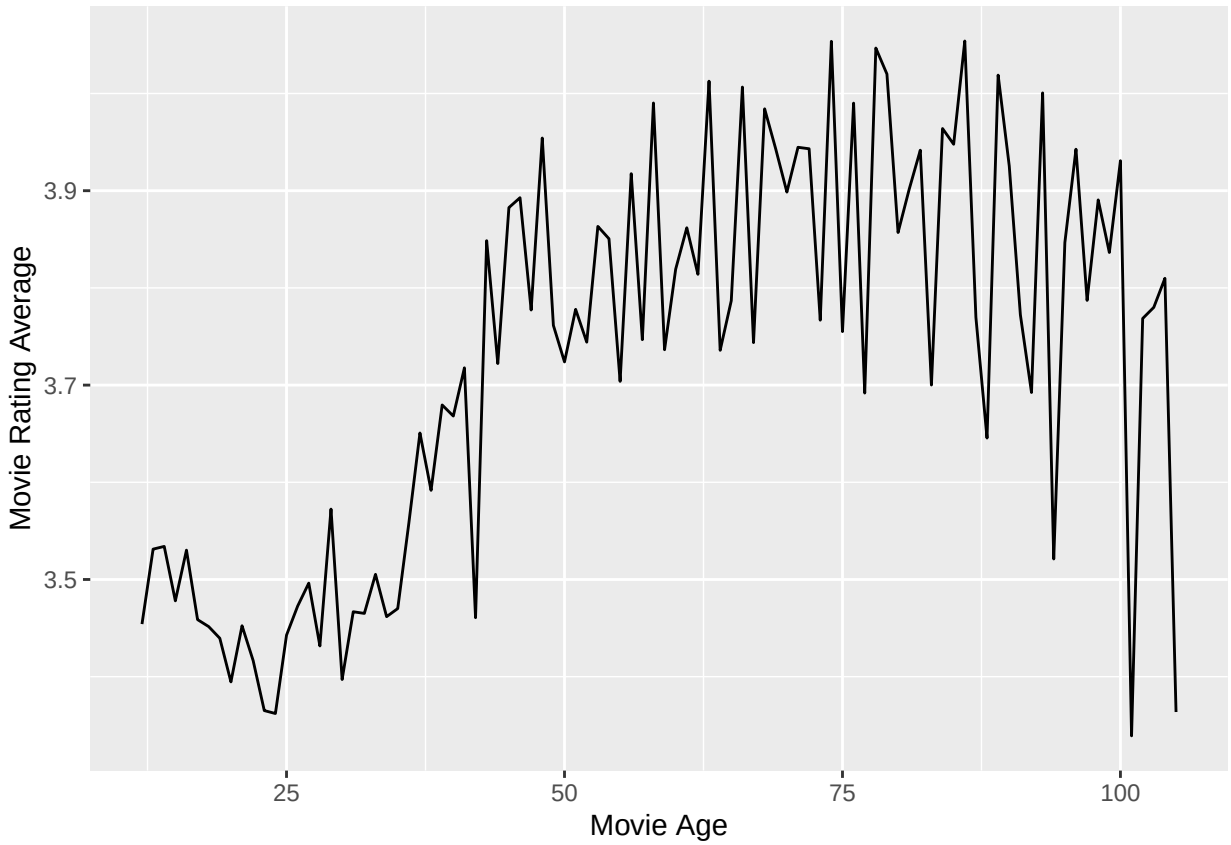
Data Exploration and Visualization

The plots in this section illustrate the impact of the movie age and genre on movie ratings.

The first plot below shows the movie rating average versus the movie age. The plot shows that there is an increase in the average movie rating as the movie age increases. However, very old movies (>100 yrs) have low ratings. Dividing movies into several bins based on their age allows for capturing the correlation between movie age and its rating.

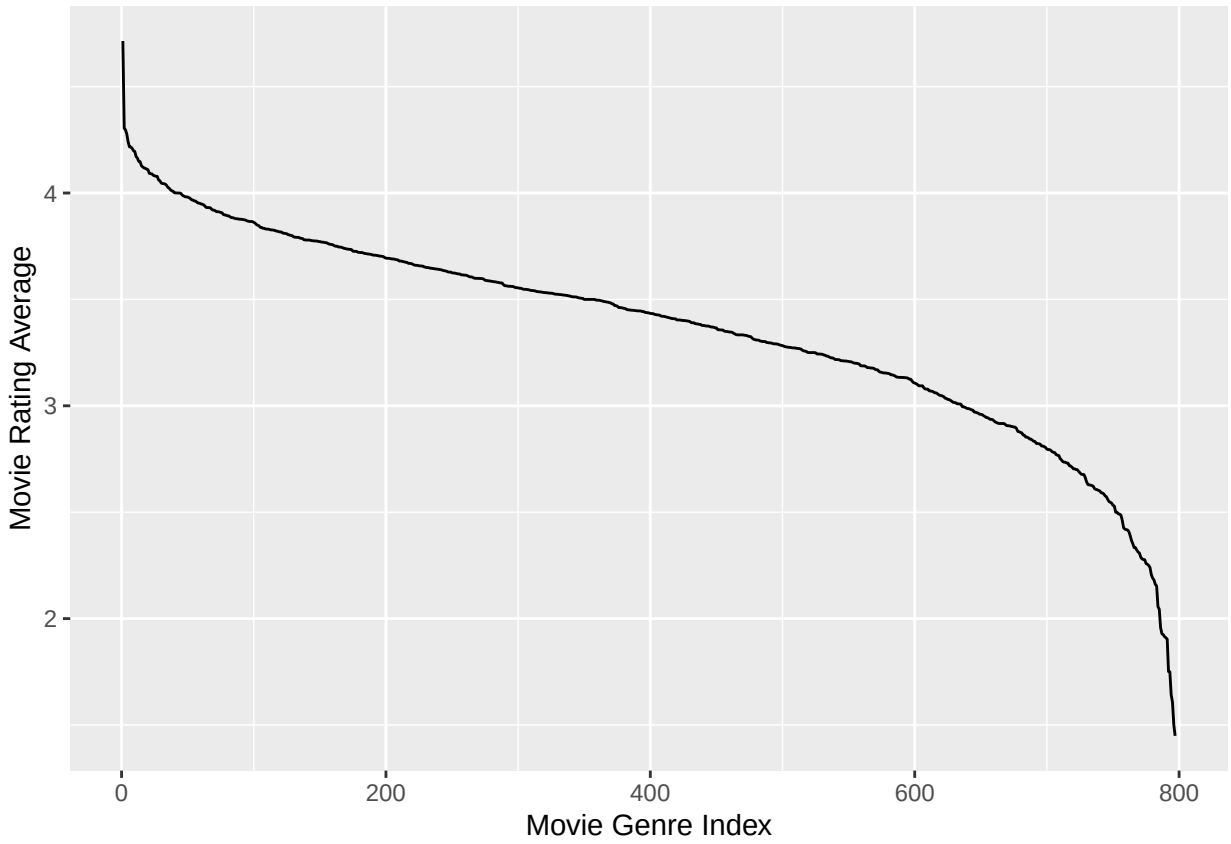
```r
# Plot avg movie rating vs movie age
movie_avg_n <- train_set %>%
  group_by(movie_age) %>%
  mutate(movie_avg = mean(rating))

movie_avg_n %>% ggplot(aes(movie_age,movie_avg)) + geom_line() +
  labs(x = "Movie Age", y = "Movie Rating Average")
```



To see how the movie genre could impact its rating, the following plot shows the movie average rating versus the genre. The plot shows clearly that movies in certain genres are rated higher. Note that each genre here can be a combination of several genres, e.g., Comedy/Musical.

```
# Explore the Movielens Dataset
# Explore Genre Effect
genre <- edx %>%
  group_by(genres) %>%
  summarize(mean_genre=mean(rating),n=n()) %>%
  arrange(-mean_genre)
# Plot avg movie rating vs movie genre
genre %>%
  ggplot(aes(1:nrow(genre),mean_genre)) +
  geom_line() +
  labs(x = "Movie Genre Index", y = "Movie Rating Average")
```



The list of genres with the highest ratings is shown below.

```
head (genre)
```

```
## # A tibble: 6 x 3
##   genres                          mean_genre     n
##   <chr>                                <dbl> <int>
## 1 Animation|IMAX|Sci-Fi                 4.71     7
## 2 Drama|Film-Noir|Romance               4.30  2989
## 3 Action|Crime|Drama|IMAX               4.30  2353
## 4 Animation|Children|Comedy|Crime       4.28  7167
## 5 Film-Noir|Mystery                     4.24  5988
## 6 Crime|Film-Noir|Mystery               4.22  4029
```

The list above shows that some genres have very few ratings. Regularization helps in offsetting the impact of the bias that these genres could cause in the recommendation algorithm results.

Building the Recommendation System

The recommendation system takes into account the genre and movie age effects in addition to the user and movie effects.

To assess the accuracy of the recommendation system, the following function calculates the root mean square error (RMSE) between predicted ratings and actual ratings.

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

As the first step in developing the recommendation system, the following piece of code calculates the average of movie ratings across all users and all movies. Here, the method predicts the same rating for all movies and all users. The average of all available ratings minimizes the least square function; Hence, it is used for prediction.

```
# Calculate the average of all movies across all users
mu_hat <- mean(train_set$rating)
```

The following calculates the RMSE of the prediction using the overall average.

```
# Calculating predicted ratings
predicted_ratings <- test_set %>%
  mutate(pred = mu_hat) %>%
  .$pred

# Calculating the model rmse
model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
print(model_1_rmse)
```

```
## [1] 1.060704
```

Movie Effect

Adding mean of the ratings of each movie to the recommendation system adds the movie effect to the prediction, i.e., how movies are rated differently from each other by users.

```
# Calculate movie effect b_i
mu <- mean(train_set$rating)
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# Calculating predicted ratings
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  .$pred
```

```
# Calculating the model rmse
model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
print(model_2_rmse)
```

```
## [1] 0.9437144
```

User Effect

To take into account the impact of movie ratings by different users, the following piece of code calculates the average of movie ratings by each user.

```
# Calaulate user effect b_u
 user_avgs <- train_set %>%
   left_join(movie_avgs, by='movieId') %>%
   group_by(userId) %>%
   summarize(b_u = mean(rating - mu - b_i))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# Calculating predicted ratings
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
```

Adding the User effect improves the RMSE:

```
# Calculating the model rmse
model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
print(model_3_rmse)
```

```
## [1] 0.8661625
```

Genre Effect

Similar to movie and user effects, the following code uses the average of movie ratings in different genres to capture how movies in various genres are rated differently.

```
# Calaulate genre effect b_g
genre_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))

# Calculating predicted ratings
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  .$pred
```

The RMSE after adding the genre effect is:

```
# Calculating the model rmse
model_4_rmse <- RMSE(predicted_ratings, test_set$rating)
print(model_4_rmse)
```

```
## [1] 0.8658242
```

Movie Age Effect

Similar to the three other predictors, the following code uses the average of movies ratings in each age bin to model the rest of variance in movie ratings.

```
# Calaulate movie age effect b_a
age_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  group_by(age_bins) %>%
  summarize(b_a = mean(rating - mu - b_i - b_u - b_g))

# Calculating predicted ratings
```

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(age_avgs, by='age_bins') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_a) %>%
  .$pred
```

```
# Calculating the model rmse
model_5_rmse <- RMSE(predicted_ratings, test_set$rating)
print(model_5_rmse)
```

```
## [1] 0.8657492
```

Comparing RMSE values above shows that adding each predictor reduces the RMSE; hence it improves the overall accuracy of the predictions made by the recommendation system.

Regularization

To penalize large estimates that come from small sample sizes, a penalty term is added to the sum of squares equation minimized. This will penalize large values of b:

$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i-b_u-b_{g_i}-b_{a_i})^2+\lambda(\sum_i(b_i^2+b_{g_i}^2+b_{a_i}^2)+\sum_u b_u^2)$

To select the optimal value of lambda, I first split the training set:

```
# Spliting training set for testing the Model Validation algorithm and selcting lamdba
valid_index <- createDataPartition(y = train_set$rating, times = 1,
                                   p = 0.2, list = FALSE)
valid_train_set <- train_set[-valid_index,]
valid_test_set <- train_set[valid_index,]

# To make sure I don't include users and movies in the valid_test set that do not
# appear in the valid_training set, I removed these using the semi_join function
valid_test_set <- valid_test_set %>%
  semi_join(valid_train_set, by = "movieId") %>%
  semi_join(valid_train_set, by = "userId")
```

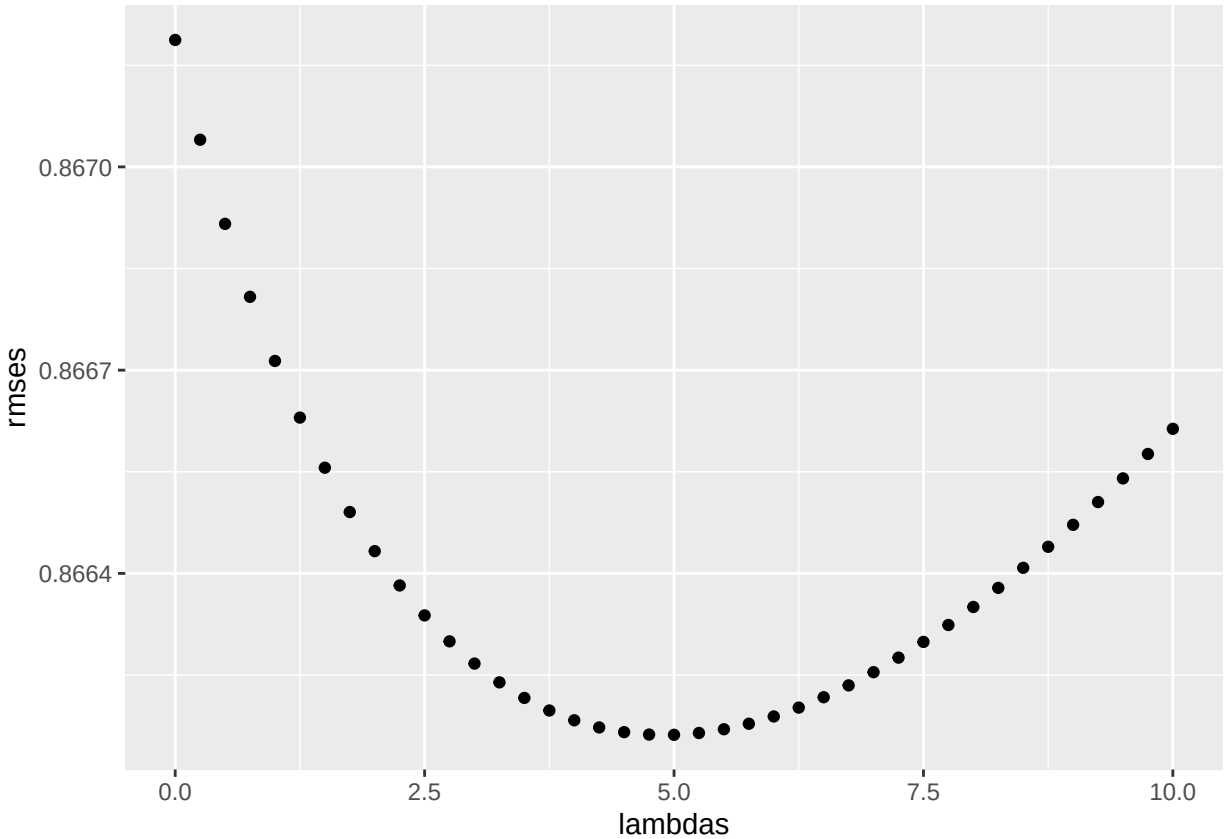
The following code calculates RMSE for several values of λ:

```
# Regularization combined movie/user/genre/movie age effect
# Choosing lambda
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(valid_train_set$rating)
  b_i <- valid_train_set %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  b_u <- valid_train_set %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  b_g <- valid_train_set %>%
    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))
```

```
  b_a <- valid_train_set %>%
    left_join(b_i, by="movieId") %>%
    left_join(b_u, by="userId") %>%
    left_join(b_g, by="genres") %>%
    group_by(age_bins) %>%
    summarize(b_a = sum(rating - b_i - b_u - b_g - mu)/(n()+l))
  predicted_ratings <-
    valid_test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_g, by = "genres") %>%
    left_join(b_a, by = "age_bins") %>%
    mutate(pred = mu + b_i + b_u + b_g + b_a) %>%
    .$pred
  return(RMSE(predicted_ratings, valid_test_set$rating))
})
```

The following plot shows RMSE versus λ:

```
qplot(lambdas, rmses)
```



The optimal value of λ and the RMSE of the recommendation system using this value is:

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

```
## [1] 5
```

```
rmses[lambdas == lambda]
```

```
## [1] 0.8661618
```

Results

In the previous section, the RMSE of predictions for the test set was calculated as the model was being developed. This section shows the model performance and results for the validation set.

Using the Recommendation System for Movie Rating Prediction for the Validation Set

The following code adds movie age and age bins to the Validation Set:

```
# Extract the movie release year from the title for Validation Set
movie_year_validation <- validation$title %>% str_extract("\\(\\d{4}\\)") %>%
  str_extract("(\\d{4})")
# Calculate the age of the movie
validation <-  mutate(validation, movie_age = 2020-as.numeric(movie_year_validation))

# Categorizing movies based on their age
validation$age_bins <- cut(validation$movie_age, breaks=c(0,20,40,60,80,105),
                           labels=c("1-20","21-40","41-60","61-80","81-105"))
```

And the following piece of code predicts movie ratings and calculates the RMSE for the Validation Set:

```
# Calculate rmse for the validation set

mu <- mean(validation$rating)
b_i <- validation %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))
b_u <- validation %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))
b_g <- validation %>%
  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()+lambda))
b_a <- validation %>%
  left_join(b_i, by="movieId") %>%
  left_join(b_u, by="userId") %>%
  left_join(b_g, by="genres") %>%
  group_by(age_bins) %>%
  summarize(b_a = sum(rating - b_i - b_u - b_g - mu)/(n()+lambda))
predicted_ratings <-
  validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_g, by = "genres") %>%
  left_join(b_a, by = "age_bins") %>%
  mutate(pred = mu + b_i + b_u + b_g + b_a) %>%
```

```
  .$pred
print('The RMSE of precidtions for the Validation set is:')
```

```
## [1] "The RMSE of precidtions for the Validation set is:"
```

```
RMSE(predicted_ratings, validation$rating)
```

```
## [1] 0.8398526
```

The final RMSE for the Validation set is 0.8399, which is below the target value of 0.8649.

Conclusion

This project builds upon the model presented in the Harvard Edx Machine Learning course. It uses the genre and movie age effects in addition to the movie and user effects. It also expands the use of regularization to the genre and movie age predictors. The final RMSE for the Validation set is 0.8399, which is below the target value of 0.8649.

Other methods and predictors can be used to expand this work. For example, it might be possible to use the number of ratings of each movie as a predictor. Matrix factorization algorithms are also powerful methods for explaining variability and structure of data and developing a recommender model. The Recommmenderlab package allows for fitting models based on matrix factorization methods.