MovieLens L Douglas

Introduction

Recommendation systems are a common machine learning task; for this capstone assignment we have been tasked to build a movie recommendation system using the MovieLens dataset. To execute this assignment, first I built the course provided movie recommendation system logic as a baseline (i.e. looking at just the average score, movie effects, movie and user effects, regularized movie effects, and finally regularized movie and (non-regularized) user effects. The course provided logic was able to recommend a movie with an root mean square error (RMSE) of 0.865127. From there I extended the system by looking into the impact of the age of the movie at the time of rating, the genre, and the amount of time a user has been rating movies. This increased the RMSE of the recommendation system to 0.864856. (A smaller RMSE means less error and thus a better recommendation system - in the context of this project, an RMSE of 1 means the recommendation system's typical error is plus or minus 1 star.)

The MovieLens dataset has 10,000,054 observations of 6 variables:

- userId: a unique identifier tracking who did the rating (69878 users)
- movieId: a unique identifier of the movie (10677 movies)
- rating: the rating given by the user (0.5 to 5)
- timestamp: when the movie was rated
- title: the title of the movie
- genres: a list of genres for the movie (between 1-6 genres with 797 different combinations)

```
##   n_users n_movies n_genres
## 1   69878    10677      797
```

Methods

The course provided data setup split the MovieLens 10M data set into 'edx' and 'validation' sets; I set the validation set aside for testing the model at the end. Looking at the edx data, there was only limited cleaning needed, where I pulled out the year the movie was rated, the year the movie was produced, and calculated how old the movie was when it was rated.

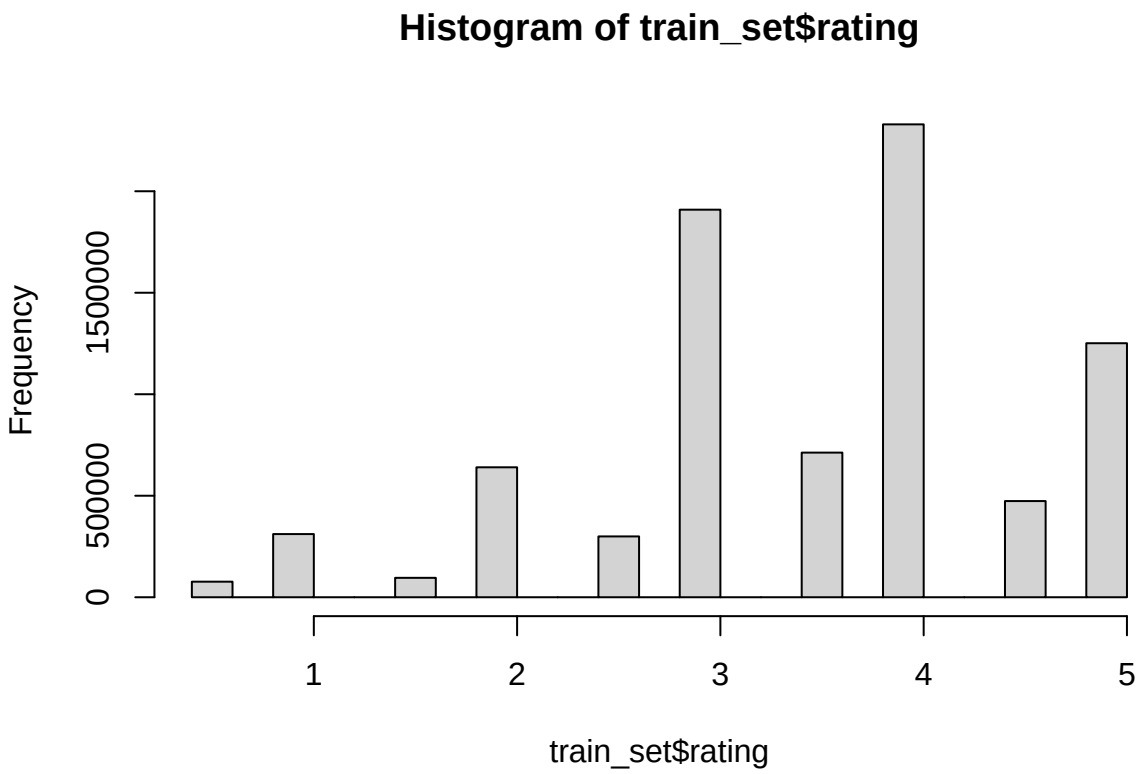
```
# create validation set with 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in 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)

# clean up the environment
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

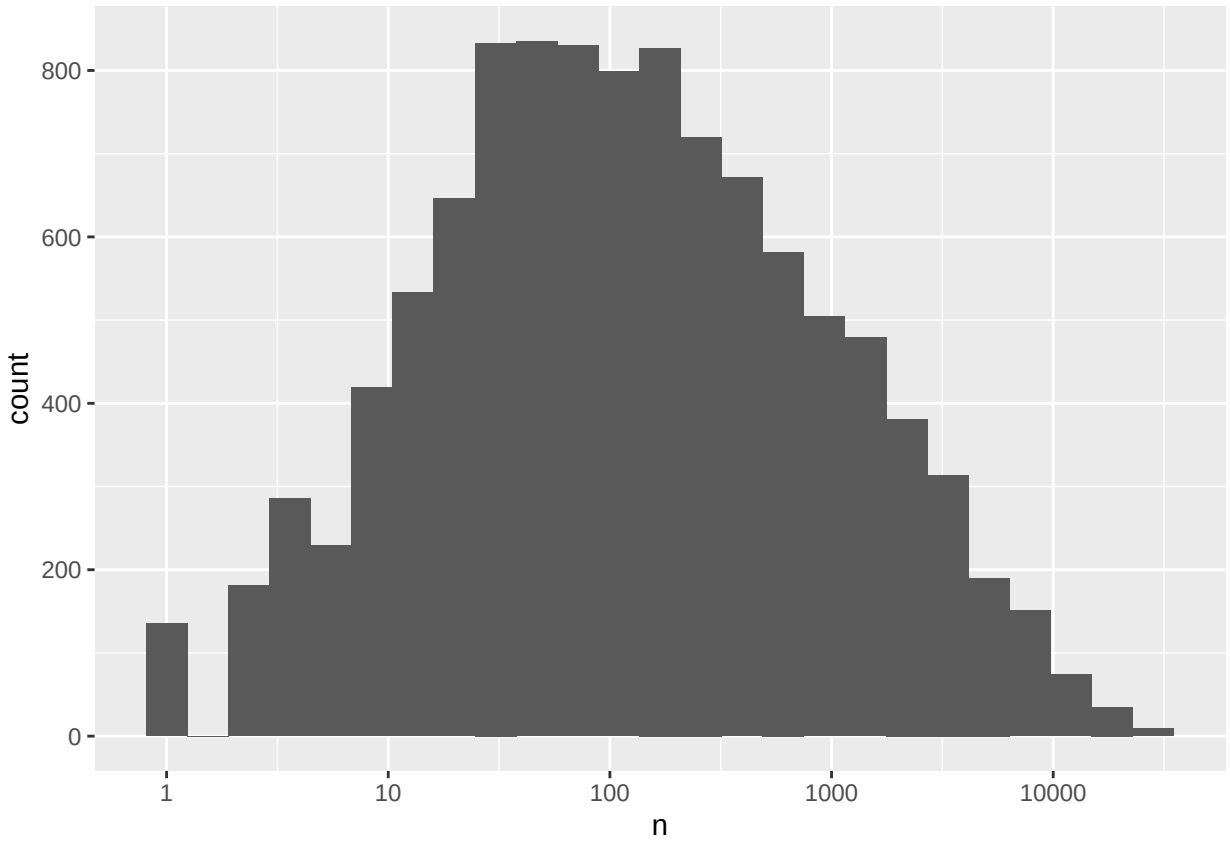
```
# clean up the data set: pull out rating year, movie year, and age at rating
edx_clean <- edx %>%
  mutate(rating_year = year(as_datetime(timestamp)),
         movie_year = str_extract(title, pattern = "(\\(\\d{4}\\))"),
         movie_year = as.numeric(str_extract(movie_year, pattern = "(\\d{4})")),
         age_at_rating = rating_year - movie_year) %>%
  filter(age_at_rating >= 0) # filter out a few with negative ages as errors in data
```

Since one shouldn't build and test their models on the validation set, I created training and test sets from the edx data. A quick histogram of the rating in the training set shows users tend to rate well (e.g. 4s) and use whole numbers more often than halves.

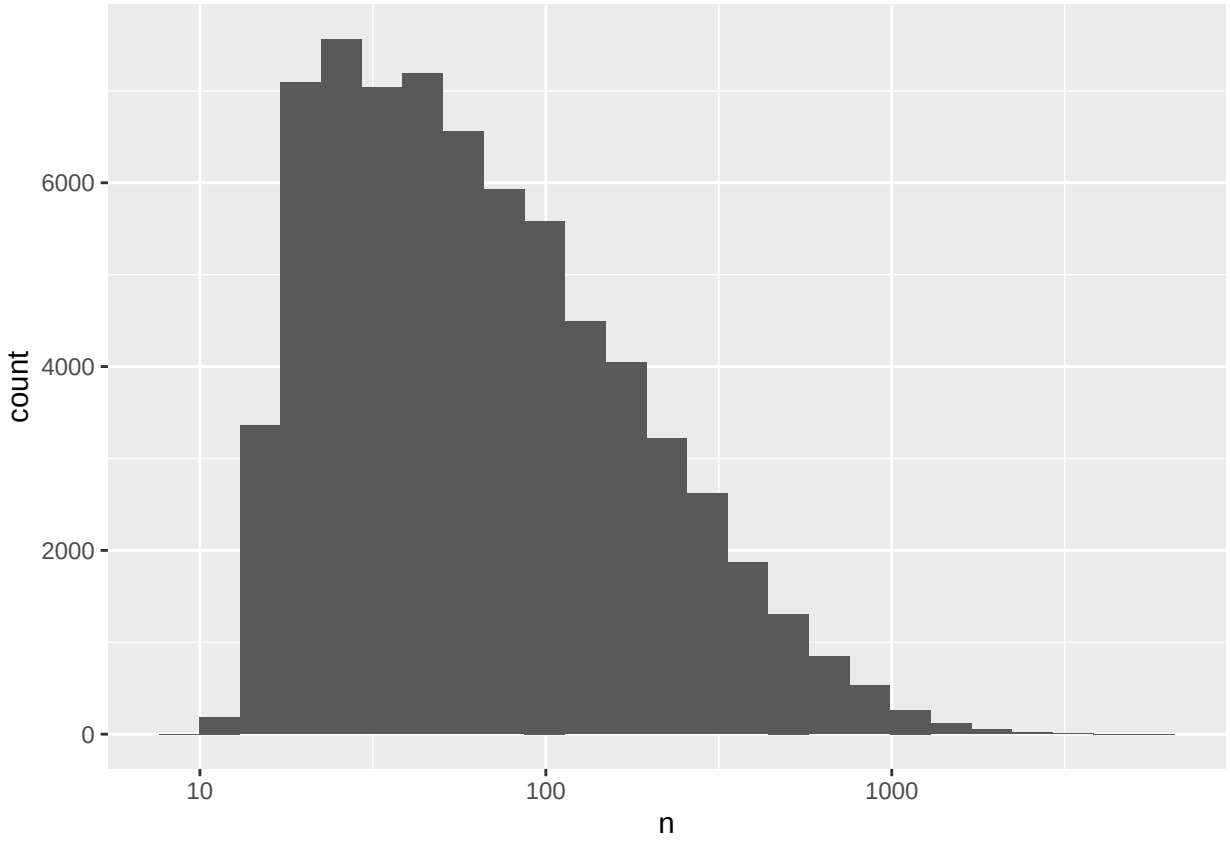
```
# build training and test sets
set.seed(1, sample.kind="Rounding")
test_index <- createDataPartition(y = edx_clean$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx_clean[-test_index,]
test_set <- edx_clean[test_index,]

# create a histogram of the training set ratings
hist(train_set$rating)
```



A histogram of movieId shows some movies are rated much more frequently than others.



And a look at the userIDs shows many people don't rate more than a hundred movies, but there is long tail of prolific raters.



Equipped with this knowledge, I then used the course provided logic and approach as a baseline. The simplest approach would be to say every movie has the same rating and any deviation is just random variation. To do this, I built a root mean square error function and used the mean of the ratings in the training set to predict each movie's rating in the test set. The RMSE isn't great:

```
# define an RMSE function to evaluate the models
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}

#### Build a model that uses the average ####

# find the mean rating of the training set
mu <- mean(train_set$rating)

# use this mean to predict the test set rating
rmse_table <- tibble(approach = "Use the average", rmse = RMSE(mu, test_set$rating))
rmse_table
```

```
## # A tibble: 1 x 2
##   approach           rmse
##   <chr>             <dbl>
## 1 Use the average 1.06030
```

I then used the same approach as the textbook, computing the least squares estimate using a prediction comprised of the the average for all movies (mu) combine with the individual movie effects for each movie

(b_i). Finding the individual movie effects for each movie simply involved finding how much each movie's rating differed from the average.

I followed the same process to add in user effects.

```
#### Build a model that uses movie effects ####

# find the average rating of each movie in the training set
movieID_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))

# join the movie averages ratings to the test set and use them to predict ratings
# in the test set
predicted_ratings <- test_set %>%
  left_join(movieID_avgs, by = 'movieId') %>%
  mutate(b_i = replace(b_i, is.na(b_i), mean(b_i, na.rm = TRUE)), #impute missing movies
                                                                  #with the mean
         pred = mu + b_i) %>%
  pull(pred)

# add the RMSE to the table
rmse_table <- bind_rows(rmse_table,
          tibble(approach = "Add movie effects",
                 rmse = RMSE(predicted_ratings, test_set$rating)))

#### Movie and User effects model ####
# find the average rating by each user in the training set
userID_avgs <- train_set %>%
  left_join(movieID_avgs, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

# join the movie averages ratings and user averages to the test set and use them to
# predict ratings in the test set
predicted_ratings <- test_set %>%
  left_join(movieID_avgs, by = 'movieId') %>%
  left_join(userID_avgs, by = 'userId') %>%
  mutate(b_i = replace(b_i, is.na(b_i), mean(b_i, na.rm = TRUE)), #impute missing movies
                                                                  #with the mean
         pred = mu + b_i + b_u) %>%
  pull(pred)

#add the RMSE to the table
rmse_table <- bind_rows(rmse_table,
                        tibble(approach = "Movie and User effects",
                               rmse = RMSE(predicted_ratings, test_set$rating)))
rmse_table
```

```
## # A tibble: 3 x 2
##   approach                  rmse
##   <chr>                    <dbl>
## 1 Use the average        1.06030
## 2 Add movie effects      0.943836
```

```
## 3 Movie and User effects 0.865264
```

Looking at these RMSEs, the prediction accuracy has greatly improved, especially with the combination of movie and user effects! This makes sense, as these are likely the two biggest drivers of a movie's rating: what movie it is an who is watchin it.

Thinking back on the histogram of movie rating frequency, there were some movies with lots of ratings and some without many at all. To remove noise from the movies with a very small number of ratings, the textbook uses regularization to filter out this noise. This is done by 'penalizing' movies that have a limited number of ratings by reducing their potential variability. Since we don't know how much to penalize these small-n movies, I work through a number of possible penalization levels ('lambdas') and select the one that results in the lowest RMSE.

```
#### regularized movie effects model ####
# create a set of potential lambdas
lambdas <- seq(0, 10, 0.25)

# create movie averages and count the number of ratings per movie
train_sums <- train_set %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())

# test the lambdas to see how well the model predicts the test set for each
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test_set %>%
    left_join(train_sums, by='movieId') %>%
    mutate(b_i = s/(n_i+l),
           b_i = replace(b_i, is.na(b_i), mean(b_i, na.rm = TRUE)), #impute missing movies
                                                                    #with the mean
           pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})

# store the best lambda (i.e. smallest RMSE) for use below
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 2.5
```

```
#use the best lambda to build the regularized movie averages
movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n())

# predict ratings in the test set using regularized movie effects
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by = 'movieId') %>%
  mutate(b_i = replace(b_i, is.na(b_i), mean(b_i, na.rm = TRUE)), # impute missing movies
                                                                  # with the mean
         pred = mu + b_i) %>%
  pull(pred)

#add the RMSE to the table
```

```
rmse_table <- bind_rows(rmse_table,
                        tibble(approach = "Regularized movie effects only",
                               rmse = RMSE(predicted_ratings, test_set$rating)))

#### Regularized movie effects and user effects ####

# predict ratings in the test set using regularized movie effects and user effects
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by = 'movieId') %>%
  left_join(userID_avgs, by = 'userId') %>%
  mutate(b_i = replace(b_i, is.na(b_i), mean(b_i, na.rm = TRUE)), # impute missing movies
                                                                  # with the mean
         pred = mu + b_i + b_u) %>%
  pull(pred)

# add the RMSE to the table
rmse_table <- bind_rows(rmse_table,
                        tibble(approach = "Regularized movie effects and user effects",
                               rmse = RMSE(predicted_ratings, test_set$rating)))
rmse_table
```

```
## # A tibble: 5 x 2
##   approach                                   rmse
##   <chr>                                     <dbl>
## 1 Use the average                          1.06030
## 2 Add movie effects                        0.943836
## 3 Movie and User effects                   0.865264
## 4 Regularized movie effects only           0.943765
## 5 Regularized movie effects and user effects 0.865127
```

Using the regularized movie effects improves the model, as can be seen by the decreased RMSE. Adding non-regularized user effects to the regularized movie effects improves things even further.

Here the course-provided logic moves into using matrix factorization, but there are other aspects of the data not yet utilized: the age of the movie when it is rated and the genres of the movie. These are included in the model using the same logic as above. The age of the movie when it is rated matters as the viewer may have excitement about a new release or be watching a 'must watch classic' - both of which would influence how a viewer perceived a movie. Genres could be important because some genres or mixes of genres might be on average more favored than others - an 'Action-Adventure' movie is a known blockbuster formula, but an 'Adventure-Drama -Romance-Sci-Fi' may be a bit too much mixed together. Pulling out the age of the movie, genres, and combined age and genre effects and adding them to our existing RMSE table:

```
## # A tibble: 8 x 2
##   approach                                                rmse
##   <chr>                                                  <dbl>
## 1 Use the average                                      1.06030
## 2 Add movie effects                                    0.943836
## 3 Movie and User effects                               0.865264
## 4 Regularized movie effects only                       0.943765
## 5 Regularized movie effects and user effects           0.865127
## 6 Regularized movie, user, and age of movie effects    0.864644
## 7 Regularized movie, user, and genre of movie effects  0.864757
## 8 Regularized movie, user, age, and genre of movie effects 0.864290
```

Continued improvement! But not as much as I would have liked - the matrix factorization used in the texbook may be a better path, as it seems like things like genre preferences may be very user specific (i.e. I may like scifi while someone else likes romcoms...)

Adding one more layer, what about maturing palettes of raters? It would make sense that as people watch more movies, they become more discerning movie raters and may rate things more critically. To do this, I find the first year each movie viewer rated a movie, then calculate the 'rater age' for each movie each when it is rated. I restricted myself to only using the training set to find the rater age to maintain similar logic to the above types of effects calculations.

```
#### Adding user age - ####
# Maybe people get harsher as they watch more movies and develop their palate?

# find the first year each user rated a movie ('start')
train_start_year <- train_set %>%
  group_by(userId) %>%
  summarize(start = min(rating_year))

# find the years that a user had been rating a movie when they made a rating ('rater_age')
train_set_plus_age <- train_set %>% left_join(train_start_year) %>%
  mutate(rater_age = rating_year - start)
```

```
## Joining, by = "userId"
```

```
# calculate 'user age' effects
rater_age_avgs <- train_set_plus_age %>%
  left_join(movie_reg_avgs, by = 'movieId') %>%
  left_join(userID_avgs, by = 'userId') %>%
  left_join(age_avgs, by = 'age_at_rating') %>%
  left_join(genre_avgs, by = 'genres') %>%
  group_by(rater_age) %>%
  summarize(b_ra = mean(rating - mu - b_i - b_u - b_a - b_g))

# add user ages to the test set
test_start_year <- test_set %>%
  group_by(userId) %>%
  summarize(start = min(rating_year)) # one could also add back in the start years
                                      # from the training set and find a 'true'
                                      # user start year without overtraining, but
                                      # they have been left separate here for
                                      # simplicity's sake
test_set_plus_age <- test_set %>% left_join(test_start_year) %>%
  mutate(rater_age = rating_year - start)
```

```
## Joining, by = "userId"
```

```
# predict test set ratings using regularized movie, user, age, genre, and rater age effects
predicted_ratings <- test_set_plus_age %>%
  left_join(movie_reg_avgs, by = 'movieId') %>%
  left_join(userID_avgs, by = 'userId') %>%
  left_join(age_avgs, by = 'age_at_rating') %>%
  left_join(genre_avgs, by = 'genres') %>%
  left_join(rater_age_avgs, by = 'rater_age') %>%
```

```
  mutate(b_i = replace(b_i, is.na(b_i), mean(b_i, na.rm = TRUE)), # impute missing movies
                                                                  # with the mean
         pred = mu + b_i + b_u + b_a + b_g + b_ra) %>%
  pull(pred)

# add the RMSE to the table
rmse_table <- bind_rows(rmse_table,
                        tibble(approach = "Regularized movie, user, age, genre, and rater age effects",
                               rmse = RMSE(predicted_ratings, test_set_plus_age$rating)))
rmse_table
```

```
## # A tibble: 9 x 2
##   approach                                                    rmse
##   <chr>                                                      <dbl>
## 1 Use the average                                          1.06030
## 2 Add movie effects                                        0.943836
## 3 Movie and User effects                                   0.865264
## 4 Regularized movie effects only                           0.943765
## 5 Regularized movie effects and user effects               0.865127
## 6 Regularized movie, user, and age of movie effects        0.864644
## 7 Regularized movie, user, and genre of movie effects      0.864757
## 8 Regularized movie, user, age, and genre of movie effects 0.864290
## 9 Regularized movie, user, age, genre, and rater age effects 0.864240
```

Results

Given the above model with regularized movie, user, age, genre, and age of rater effects, how do things go with the validation set?

```
#### validation - how'd we do? ####
# pre-process validation set (adding movie age, remove errors)
validation_clean <- validation %>%
  mutate(rating_year = year(as_datetime(timestamp)),
         movie_year = str_extract(title, pattern = "(\\(\\d{4}\\))"),
         movie_year = as.numeric(str_extract(movie_year, pattern = "(\\d{4})")),
         age_at_rating = rating_year - movie_year) %>%
  filter(age_at_rating >= 0)

# calculate user ages for the validation set
validation_start_year <- validation_clean %>% group_by(userId) %>%
  summarize(start = min(rating_year))
validation_set_plus_age <- validation_clean %>% left_join(validation_start_year) %>%
  mutate(rater_age = rating_year - start)
```

```
## Joining, by = "userId"
```

```
# predict ratings in the validation set using the final model logic in methods section
predicted_ratings <- validation_set_plus_age %>%
  left_join(movie_reg_avgs, by = 'movieId') %>%
  left_join(userID_avgs, by = 'userId') %>%
  left_join(age_avgs, by = 'age_at_rating') %>%
```

```
  left_join(genre_avgs, by = 'genres') %>%
  left_join(rater_age_avgs, by = 'rater_age') %>%
  mutate(b_i = replace(b_i, is.na(b_i), mean(b_i, na.rm = TRUE)), # impute missing movies
                                                                  # with the mean
         pred = mu + b_i + b_u + b_a + b_g + b_ra) %>%
  pull(pred)

# add the RMSE to the table
rmse_table <- bind_rows(rmse_table,
                        tibble(approach = "vs. Validation set",
                               rmse = RMSE(predicted_ratings,
                                           validation_set_plus_age$rating)))
rmse_table
```

```
## # A tibble: 10 x 2
##    approach                                                    rmse
##    <chr>                                                      <dbl>
##  1 Use the average                                          1.06030
##  2 Add movie effects                                        0.943836
##  3 Movie and User effects                                   0.865264
##  4 Regularized movie effects only                           0.943765
##  5 Regularized movie effects and user effects               0.865127
##  6 Regularized movie, user, and age of movie effects        0.864644
##  7 Regularized movie, user, and genre of movie effects      0.864757
##  8 Regularized movie, user, age, and genre of movie effects 0.864290
##  9 Regularized movie, user, age, genre, and rater age effects 0.864240
## 10 vs. Validation set                                       0.864856
```

Pretty good! The model beats the course-recommended target of an RMSE less than 0.86490, so I'm happy with the result.

Conclusion

Building a good recommendation system is really an exercise in understanding the people for whom the system is supposed to build recommendations. With that in mind, the things I modeled matched what would come up in common conversation about movie recommendations among non-data scientists: average movie ratings, what movie it was, who was rating it (and how long they've been rating movies), how old the movie was, and what genre it was. (Along the way, I also used regularization to reduce some variability of infrequently rated movies; but even this would make sense to a non-data scientist - it makes intuitive sense that you can't completely trust a rating by only one or a handful of people!)

To improve the system, we would simply continue our discussion with our non-data scientist friend and build tools to implement the ideas that came up. I could capture the fact that different people like different types of movies (i.e. use matrix factorization like the textbook). I could also build additional predictors that capture other elements that influence people's ratings: - What the rating of the movie is when someone watches it (Could this anchor their rating?) - When someone is rating a movie compared to other users (Are the first raters more tentative about giving five stars since they don't want to be 'wrong'? Or more excited about a new movie and more likely to rate higher?) - How many people have rated a movie when it is watched and rated (Do people follow the crowd? Or are they contrarian when something gets 'too popular')

In the end, while interesting, at some point increased accuracy doesn't really contribute to the experience of users - if a movie service like Netflix can get within .8 or .3 in their guessed ratings for users, does it really matter? As long as the system close by a star or so, users will be happy!