EdxDataScienceCapstoneMovieRecommender

John Wallace

March 28, 2019

Executive Summary

This report has been created in partial fulfillment of the requirements for the Harvard EdX Data Science Program Certificate Capstone. This report details a movie recommender system that was created based on the data from the GroupLens research Lab database. The original database contains over 20 Million ratings for over 27,000 movies by more than 138,000 users. The analysis uses a subset of the data (the m1-10m.zip file) which was downloaded from the Grouplens web site. Root Mean Square Error (RMSE) will be used as the metric to determine the success of the model. The goal of this project is to create a recommender system that is optimized for RMSE and has an RMSE of < 0.87916 or better. The methodology followed to develop the model is based on the process as described by Dr. Rafael A. Irizarry in the book Introduction to Data Science. This method utilizes a prediction model based on the equation Y[u,i] = mu + b_i + b_u where Y[u,i] is the rating for movie i by user u, mu is the average movie rating, b_u is the user specific effect, b_i is a movie specific effect. R script was provided to download and extract a training and validation data set. Details of the data set are described below. The model was developed based on the training set (refered to as edx in the code) and tested against the validation data set (refered to as validation). The model performed well with an **RMSE of 0.86522** which is very good with respect to the defined rubric *(25 points: RMSE <- 0.87750).*

Note that other model creation methods (namely recommenderlab) were explored but were not able to be utilized due to limted memory size on the PC and the size of the edx dataset. In order to familiarize myself with how the recommenderlab package could have been used, a smaller dataset consisting of a subset of the validation data only was utilzied to create and test a recommender. The analyis is presented in an Appendx as reference.

Introduction

Recommender systems are utilized in many areas to make specific recommendations to users. Amazon is a common example where products are recommended based on a users purchase history or similarity of items purchased to other items. Another famous example is the Netflix challenge in which teams were challenged to improve on their movie recommendation systems.

There are many types of recommendation systems but one popular approach is to utilize a collaborative filtering approach. The collaborative filtering algorithm can be either user based (UBCF) or item based (IBCF). For UBCF the ratings users gave movies are compared to estabilish how similar they are in terms of rating movies and items are recommended by finding similar users to the user we would like to recommend a move to. For IBCF, items are recommended that are similar to what a user has given high ratings to in the past.

As noted above, the methodology utilzed to create the recommender system is very similar to the methodology as defined by Dr. Irizarry in Chapter 34.7 of his book as referenced above.

The following steps were used to develop and test the model:

1. Import the movielens dataset
2. Create a training dataset and validation dataset from the original data
3. Perform data cleaning by checking for NA's and NULL's
4. Perform exploration and visualization of the dataset to understand details of the data
5. Create the model on the training set

6. Utilize the validation set to calculate RMSE

Note that dslabs, tidyverse and caret libraries were used extensively.

Note that after downloading the data, the training (edx) & validation datasets were written to a local file to facilitate exploration of the data in different sessions without having to download the files again. (If the reader choses to run the script, the download code that is currently commented out will need to be uncommented and executed to download the datafile and create the edx & validation data sets)

Methods and Analysis

This section details the analysis and method used to develop the model. Each of the steps outlined in the Introduction section above is explained in detail.

Step 1: Import the movielens data & create a training dataset and validation dataset from the original data

The code provided from the capstone instructions was executed to download and generate the data sets. Two data sets were generated; a training set, called edx and a validation set called validation. In order to expidite the development of the model without reloading the data sets each time a new session is started, the training and validation data sets were saved to local files using the write.csv command. Subsequent sessions simply loaded the data set using the load.csv command.

Step 2: Perform data cleaning by checking for NA's and NULL's

Next the data was checked to see if there were any NA's or NULL's which might create problems when developing the model.

```
# Do we have any NA's in the data?
edxNA <- apply(edx, 2, function(x) any(is.na(x)))
edxNA
```

```
##    userId   movieId    rating timestamp     title    genres
##     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE
```

```
# There are no NA's (Note you would use colSums(is.na(df) and which(is.na(df))) to see which
#  entries were NA's

# Check for NULL's
edxNULL <- apply(edx, 2, function(x) any(is.null(x)))
edxNULL
```

```
##    userId   movieId    rating timestamp     title    genres
##     FALSE     FALSE     FALSE     FALSE     FALSE     FALSE
```

```
# there are no NULL's
#
# End Data Cleaning
#
```

Step 3: Perform exploration and visualization of the dataset to understand details of the data

In this step, we determine the type and quantity of data available.

```
typeof(edx)
```

```
## [1] "list"
```

```
summary.default(edx)
```

```
##           Length  Class  Mode
## userId    9000055 -none- numeric
## movieId   9000055 -none- numeric
## rating    9000055 -none- numeric
## timestamp 9000055 -none- numeric
## title     9000055 -none- character
## genres    9000055 -none- character
```

```
summary.default(validation)
```

```
##           Length Class  Mode
## userId    999999 -none- numeric
## movieId   999999 -none- numeric
## rating    999999 -none- numeric
## timestamp 999999 -none- numeric
## title     999999 -none- character
## genres    999999 -none- character
```

Note that edx is about 9M rows and validation is about 999K rows.
The data is a list with 6 fields. The key fields we are interested in are the userId, movieId, and the rating (a numeric value from 1 to 5 with 5 representing a high rating). These fields will be utilized as the model is developed with the goal being to use the userId and movieId to estimate a rating for a given movie.

Next, determine how many movies & users are in the training data set

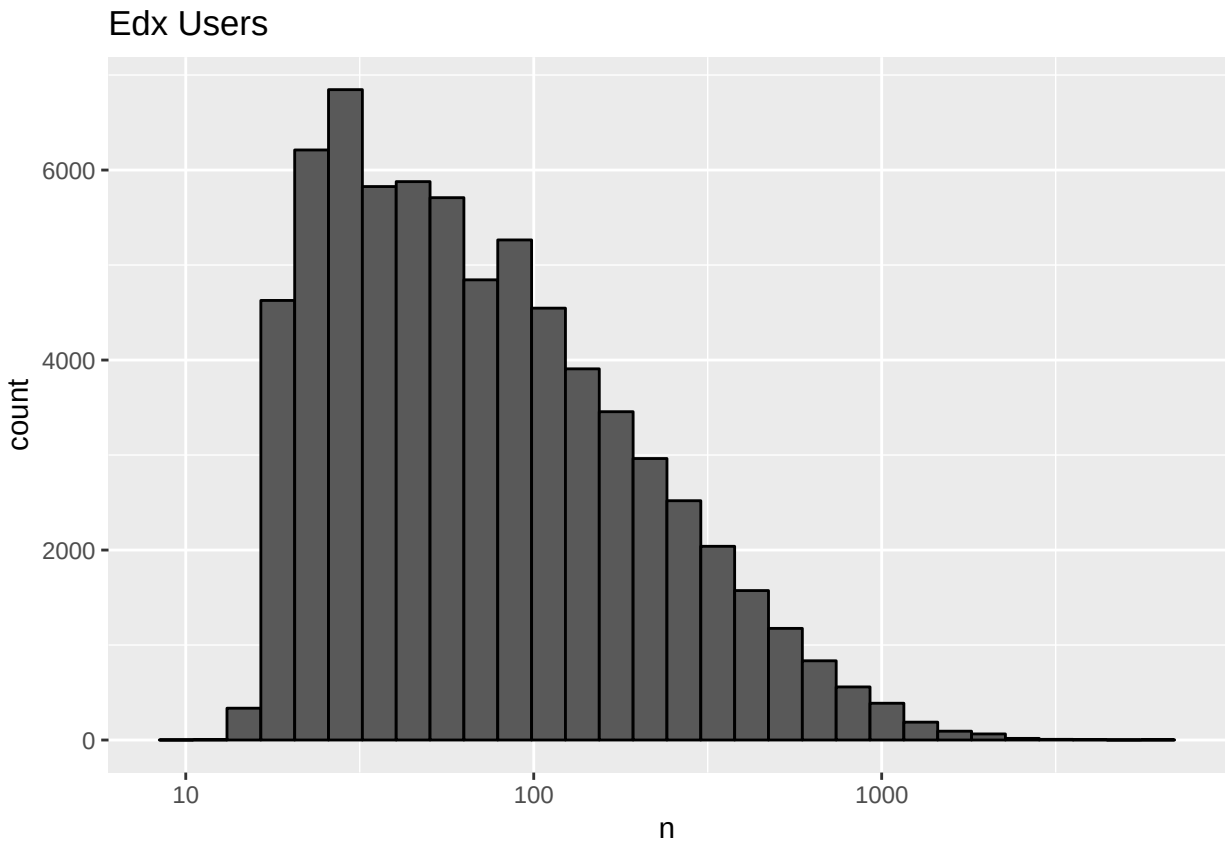
```
edx %>%
  summarize(n_users = n_distinct(userId),
            n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

```
# 69,878 users and 10,677 movies in edx
```

Examine the distribution of users in the training data

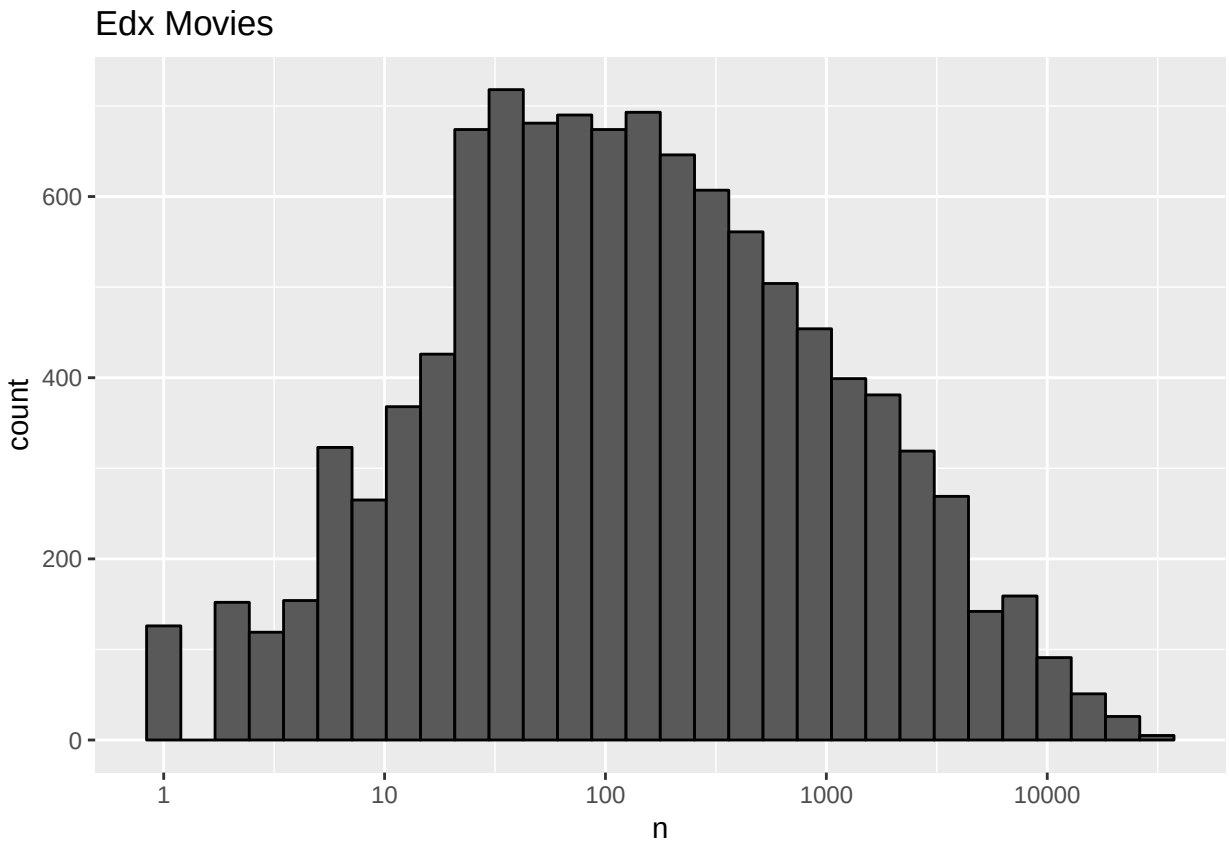
```
edx %>%
  count(userId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Edx Users")
```



Note that some users only have a few ratings; others have a large number of ratings.

Also, examine the distribution of movies in the training data.

```
edx %>%
  count(movieId) %>%
  ggplot(aes(n)) +
  geom_histogram(bins = 30, color = "black") +
  scale_x_log10() +
  ggtitle("Edx Movies")
```



Some movies have lots of ratings and some only a few. Most have between 300 and 600 ratings.

At this point, there is a good understanding of the type and structure of the data and the distribution of movies and users.

Step 4: Create the model on the training set

The next step is to create the model using the training data (edx).

Note that the rubric for the project stated that the edx data set should be used for the training and the validation set should be used for the final RMSE calculations to see how good the model performs. First, setup a function to be used to calculate the RMSE and use as a loss function. This function takes 2 variables; the actual ratings (from the data) and the predicted ratings. It then uses the formula Sqrt(sum(actual-predicted)^2/N) to calculate the RMSE and returns this value.

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

As noted above, the development methodology is very similar to the process presented in Dr. Irizarry's book Introduction to Data Science. While many of the steps could have been combined to produce the final model, I have chosen to develop the model incrementally using the following steps:

1. Develop a model based only on the average rating of the entire movie set
2. Add a term, b_i that is based on the average rating of each movie
3. Add a term, b_u that is based on the average rating by each user
4. Apply regularization to adjust for movies that are only rated by a few users

These steps are outlined as follows:

1. Develop a model based only on the average rating of the entire movie set

Start with a model which assumes y[u,i]= mu + esp[u,i] where mu is the average rating of the movies and and esp[u,i] are independent errors sampled from the same distribution centered at 0. This method just uses the average rating across all movies. We will calculate mu from the training data and apply this as the rating for all movies then calculate the RMSE this model. A data frame is also craeted to hold the comparisons for the different iterations of the model.

```
mu_hat <- mean(edx$rating)
mu_hat
```

```
## [1] 3.512465
```

```
# The average rating is about 3.5

# Predict all ratings with just the average and see what the RMSE is
Model1_rmse <- RMSE(edx$rating, mu_hat)

# General note on results:  Note that the RMSE returns a number that represents the
#   RMSE.  The range of the rating is from 1 to 5 so an RMSE of 1 represents a pretty
#   large potential error (1 star)

# Create a results table so we can track performance of differnet models
rmse_results <- data_frame(method = "Model1: Average only", RMSE = Model1_rmse)
# and check the results
rmse_results %>% knitr::kable()
```

method	RMSE
Model1: Average only	1.060331

Note that the average ratings is about 3.5 and the RMSE for Model 1 is about 1.06. General note on results: The RMSE function returns a number that represents the RMSE. The range of the rating is from 1 to 5 so an RMSE of 1 represents a pretty large potential error (1 star).

2. Add a term, b_i that is based on the average rating of each movie

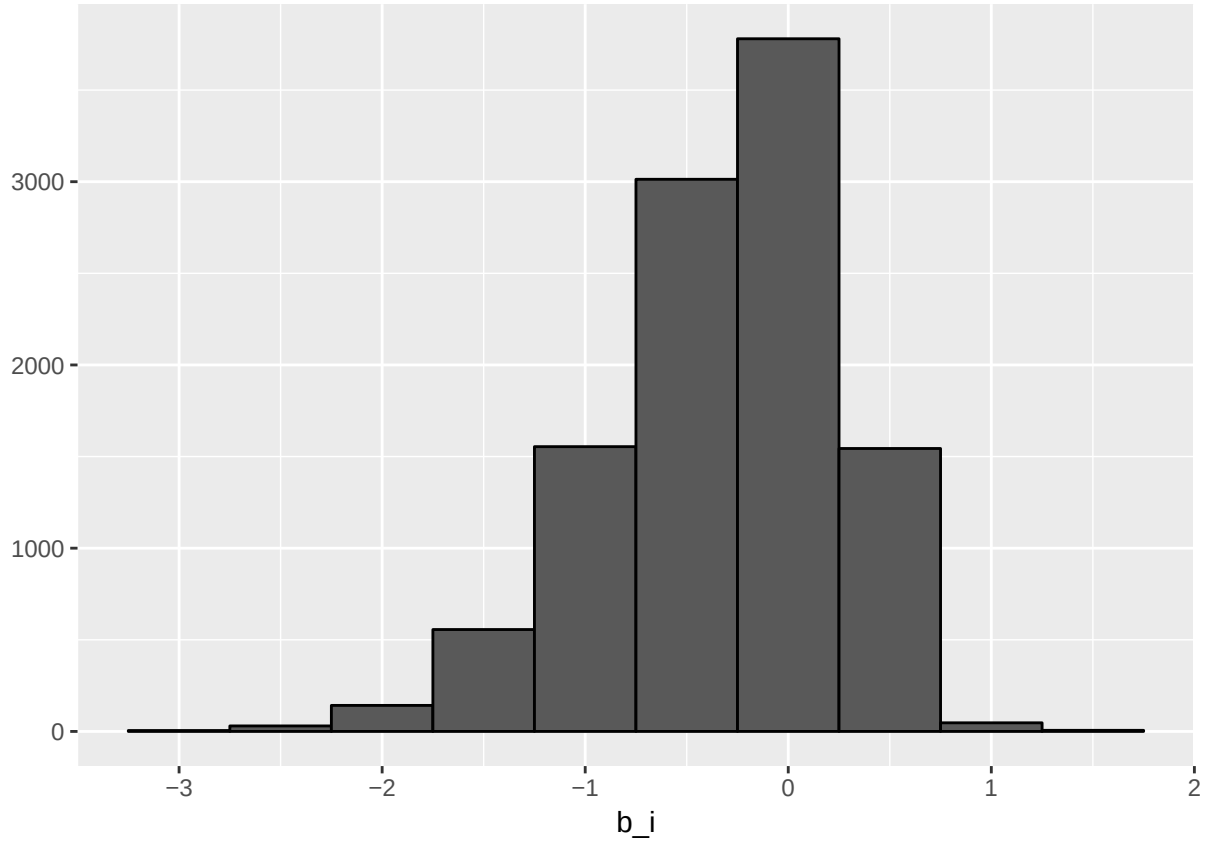
Next, note that some movies generally are rated higher than others. We can take this into account by adding a term b_i for average movie ranking for movie i. Note that the b_i is for different for each movie and represents the average rating given for that movie.

As noted in the class and book, one way to calculate b_i is to use lm to fit a linear model (using lm), but due to the size of the data set (1000's of movies) this would take a very long time to do this calculation on a PC. The R script accompanying this report shows how the lm function could be used to fit the model. One way to estimate the value b_i (wihtout using lm) would be to use the the assumption that Y[u,i] = mu_hat + b_i_hat and solve for b_i_hat = Y[u,i] - mu_hat (where Y[u,i] is the actual rating).

```
# Now create a dataframe movie_avgs that has a value (one value for each movie) which represents
#   on average how far above or below the overall mean a particular movie is. This will be
#   b_i for each movie
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))
```

We can look at the distribution of the "difference" (b_i) and we see that the range is from -3 to +1.5 which (when the average mu_hat is added) represents a rating of 1.5 to 4.5 . As expected, most movies are around 0 (which represents the overall average)

```
movie_avgs %>% qplot(b_i, geom ="histogram", bins = 10, data = ., color = I("black"))
```



This term can now be added to our original model and check the RMSE.

```
predicted_ratings <- mu_hat + edx %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

# Now check the RMSE with the new model which incorporates the movie effect
Model2_rmse <- RMSE(predicted_ratings, edx$rating)

# Add this model to our results
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model2: Movie Effect Model",
                                     RMSE = Model2_rmse ))
rmse_results %>% knitr::kable()
```

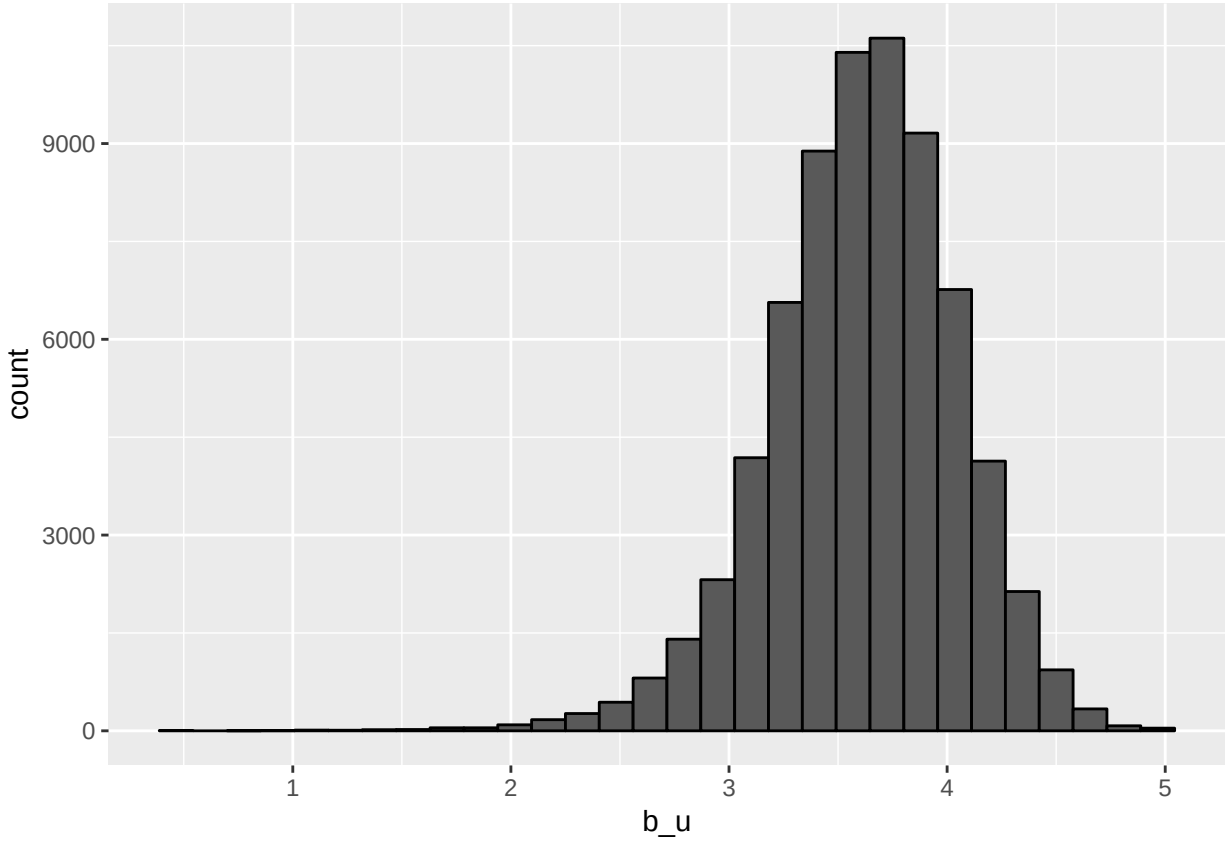
method	RMSE
Model1: Average only	1.0603313
Model2: Movie Effect Model	0.9423475

The RMSE improves to about 0.9423.

3. Add a term, b_u that is based on the average rating by each user

Next, a term will be added that represents the user effect. The methodology is similar to what was used to model the movie effect. Start by looking at the distribution of the average rating of users who have rated over 100 movies. Note that b_u is a vector where each element represents the average rating the user_id gave movies. It is shown that there is some variability across users as the distribution ranges from about 1.5 to 4.5 so adding a user term to the model should help.

```
edx %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, color = "black")
```



The new model will have both a user and movie term can be written as Y[u,i] = mu + b_i + b_u + exp[u,i] where b_u is the user specific effect. Similar to the above statement related to fitting a linear model using the lm function, this could be done, but it would take a long time to run (See the code for the example of how to use lm for this). Again, it is possible to estimate the user factor by using the equation Y[u,i] = mu_hat + b_i + b_u and rewrite as b_u = Y[u,i] - mu_hat - b.

Now calulate the new term for the user effect (b_u) and add it to the model and check the result.

```
# user_avgs will contain a term, b_u which represents the user average for each movie
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))

# Use the new model now to predict the ratings using both a user effect & a movie effect
```

```r
predicted_ratings <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  .$pred

# and  check the RMSE using the new preditions

Model3_rmse <- RMSE(predicted_ratings, edx$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model3: Movie + User Effects Model",
                                     RMSE = Model3_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Model1: Average only	1.0603313
Model2: Movie Effect Model	0.9423475
Model3: Movie + User Effects Model	0.8567039

This yields an improvement in RMSE to approximately 0.8567.

4. Apply regularization to adjust for movies that are only rated by a few users

Note that if a movie does not have a lot of ratings, there is more uncertainity and those users that do rate the movie can have an outsized influence on the rating versus other movies and larger estimates of b_i (+ or -) are more likely. In order to compensate for this (movies with few ratings) we can use a technique called regularization, which effectively takes into account the number of ratings a movie has versus other movies. Regularization changes the minimization of the least square equation by adding a penalty term. Reference the book chapter on Regularization for details. The technique adds a penalty term which includes a "tuned" variable (lambda) and the count of the number of ratings a movie has. Lambda is a tuning parameter so we can setup a loop to determine the optimium value for lambda (the value that provides the lowest RMSE) and plot the RMSE versus lambda.

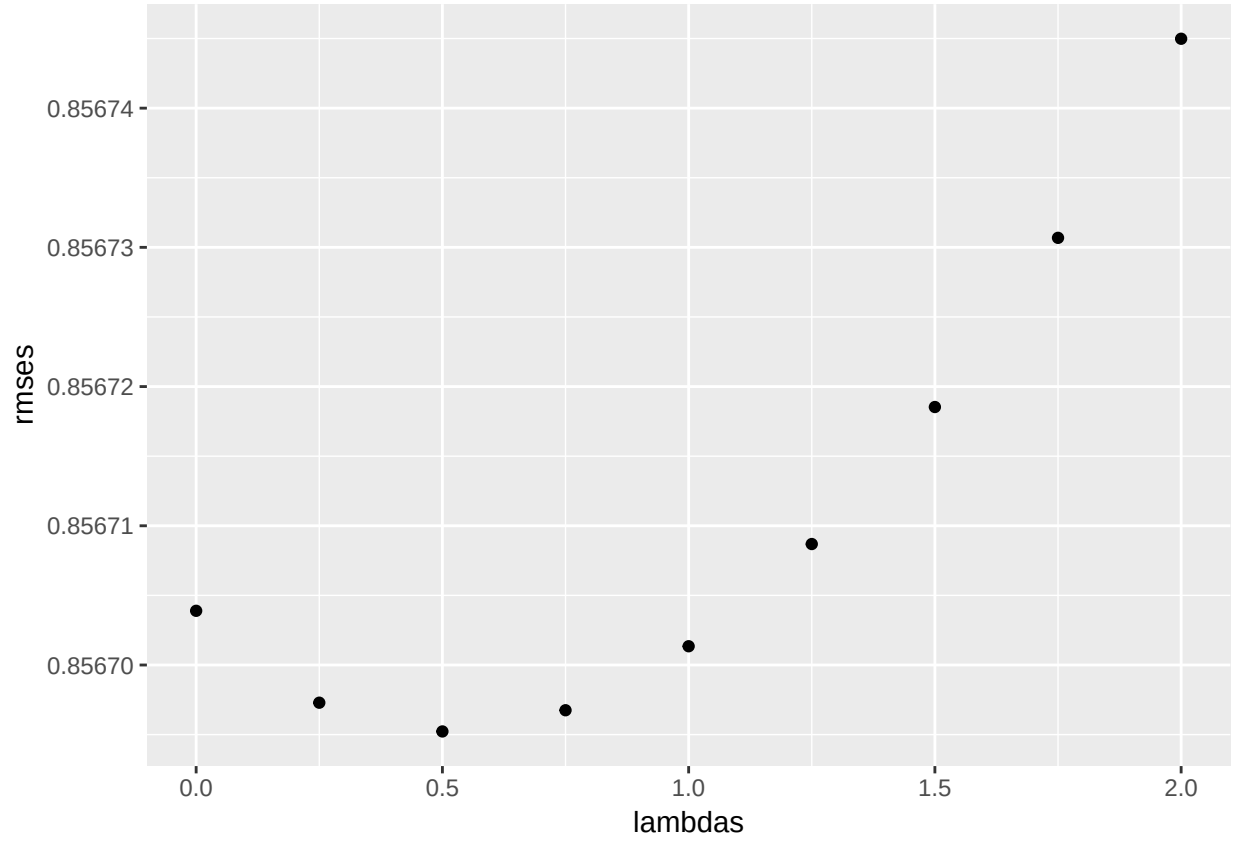
```r
# The following code runs iterations to determine the tuning parameter lambda.
# Note: Originally the sequence ran to 10 but after running the code I found that
#   the best value for Lambda was around .5 so I changed the sequence to run to 2
#   for subsequent runs in order to reduce the time to run
#
lambdas <- seq(0, 2, 0.25)

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

  mu <- mean(edx$rating)
# Calculate movie term b_i with the penality count (n)lambda
  b_i <- edx %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
# Calculate the user term b_u with the penality count (n) and lambda
  b_u <- edx %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
# Determine the predicted ratings using lambda
```

```
  predicted_ratings <-
    edx %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)
  # and return the RMSE for each lambda
  return(RMSE(predicted_ratings, edx$rating))
})

# and plot the rmse versus the lambda to determine the best tuned value
qplot(lambdas, rmses)
```




```
# from plot it looks like the best will be with lambda somewhere between 0.25 and 2.
#  So modify the seq going forward to reduce the time.  Note this was already done in
#  the code above as the sequence originally went to 10
```

and now determine the lambda which results in the lowests RMSE and utilize it in the new model.

```
# Determine which lambda produced the lowest rmse
lambda <- lambdas[which.min(rmses)]
lambda
```

```
## [1] 0.5
```

```
# a lambda of 0.5 produced the lowest rmse use this to complete the model
#
# Note that this represents the lambda (regularization) that results in the best
```

```
#   (lowest RMSE) calculated
# Add to the results for comparison
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User Effect Model",
                                     RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Model1: Average only	1.0603313
Model2: Movie Effect Model	0.9423475
Model3: Movie + User Effects Model	0.8567039
Regularized Movie + User Effect Model	0.8566952

Using a lambda of 0.5 and regularization improves the RMSE only marginally (0.856703 to 0.856695).

The final model based on the edx dataset is now complete. We need to recalculate the b_i, b_u with the best lambda as found above which will be used to test the validation model and save the model parameters.

```
l <- lambda
mu <- mean(edx$rating)
# b_i will contain movie factor at the tuned lambda value found above
b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+l))
# b_u is the user factor at the tuned lambda value found above
b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+l))

predicted_ratings <-
  edx %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
# Calculate the complete model using the tuned value of lambda.  Note this
#   is still on the training (edx) data.
Model5_rmse <- RMSE(predicted_ratings, edx$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Model5: Regularization Movie + User",
                                     RMSE = Model5_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Model1: Average only	1.0603313
Model2: Movie Effect Model	0.9423475
Model3: Movie + User Effects Model	0.8567039
Regularized Movie + User Effect Model	0.8566952
Model5: Regularization Movie + User	0.8566952

The model is now completed and we can see how it performs against the validation data set.

Step 5: Utilize the validation set to calculate RMSE

Now setup the final regularized model to use with the validation data. Utilize the value of mu, b_i and b_u found above. Define the model parameters based on the best model which is the Regularized Model with movie and user effects and add these model parameters to the validation data set (create a new data set called ModelValidation).

```
Model_mu <- mean(edx$rating)
Model_b_i <- b_i
Model_b_u <- b_u
# add the model parameters to the validation data set (create a new data set)
ModelValidation <- validation %>% left_join(Model_b_i, by = "movieId") %>%
  left_join(Model_b_u, by = "userId")
# Model Validation now contains the original validation data + the model's b_i and b_u
```

With the model parameters added to the validation data set, the model is used to predict the ratings for the validation dataset.

```
Validation_predicted_ratings <- ModelValidation %>%
  mutate(pred = Model_mu + b_i + b_u) %>% pull(pred)
```

Results

Now that the model has been created, run the predictions and review the results.

```
# And check the performance by calculation the RMSE of the predicted ratings versus the
#   actual validation ratings.
Validation_rmse <- RMSE(Validation_predicted_ratings, validation$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Validation: ",
                                     RMSE = Validation_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Model1: Average only	1.0603313
Model2: Movie Effect Model	0.9423475
Model3: Movie + User Effects Model	0.8567039
Regularized Movie + User Effect Model	0.8566952
Model5: Regularization Movie + User	0.8566952
Validation:	0.8652226

The RMSE for the Validation data set with the final model is 0.86522 which is not as quite as good as for the edx (training) data (which is as expected). These results are still much better than the initial average only, user based and movie based models for the original edx training data set.

Conclusions

The final model Y[u,i] = mu_hat + b_i + b_u where mu_hat is the overall average rating for all movies, b_i is the average for a particular movie and b_u which is the average for a particular movie represents a reasonable model to predict ratings. The model was created based on the edx training set and validated using the validation set as defined in the Capstone requirements using the given R script (to download and setup the edx and validation data sets). **The overall RMSE for the validation set is 0.86522 which is below the threshold noted in the Rubric to be awarded 25 points for the model.** Interestingly,

as noted in step 4, the addition of a regularization term only marginally improved the RMSE using the edx (training) data set. The regularization term was included in the final model but it would be interesting as a follow on project to determine if the regularization term also has only a marginal effect on the validation data as well.

Appendix A- Recommenderlab analysis

As noted, due to limited memory on my PC the full edx train set could not be utlized with the recommender lab package. In order to understand how the package could be used, I decided to use the smaller validation data set with the recommender package tocreate an UBCF and IBCF recommendation system and check the RMSE. Note that the recommenderlab pacakge was created by Michael Hahsler for use in creating and testing recommendation systems.

More information can be found at https://cran.r-project.org/web/packages/recommenderlab/vignettes/recommenderlab.pdf

Cited as Michael Hahsler (2019). recommenderlab: Lab for Developing and Testing Recommender Algorithms. R pacakge version 0.2-4. https://github.com/mhahsler/recommenderlab

The following elements of the recommenderlab package were used:

1. evaluationScheme- Creates an evaluationScheme object from a dataset. Manages the splitting of data into training & test data.
2. Recommender- Learns a model from data
3. predict- given a model (from recommender) and a new dataset, creates predictions for the new data set. A number of different prediction types (topN list, full ratings) are supported getdata- used to extract data from teh evaluationScheme object

Note that the pacakge makes use of a ratingsMatrix or realratingsMatrix type which are sparse matrices with users as the rows and item ID's (in our case, movies) as the columns.

The validation data set (which is a smaller data set than edx) was utilized to develop, create and test the models.

After loading the validation data, exploring and insuring that the data is appropriate for modeling, the following steps were taken to create and test the model

1. Wrangle the data to only have the fields of interest (user, movie & rating)
2. Convert the data into a realratingsMatrix
3. Setup an evaluationScheme with the data
4. Create the model using Recommender (note we will create both UBCF and IBCF models)
5. Create the predictions using predict
6. Evaluate the results using calcPredictionAccuracy
7. Display the RMSE

Two different models were created, the first based on an user based collaborative filter (UBCF) and the second based on an item (movie) based collaborative filter (ICBF). The models were developed on a training set of data, predictions made on a test set and the RMSE for both models compared. The UBCF model performed best with an RMSE of 1.03226 while the item based RMSE was 1.3199.

The code is provided in the accompanying R file but is not included in this document.