MovieLens Report

Spencer Fisher

12/20/2020

Introduction

I created a machine learning model that predicts the rating that an individual user will give a certain movie. This is useful because models like this can be used to recommend movies to a user that they would like to watch based on the rating of previous movies that they have watched. Better recommendations can lead to more viewing time by users and higher customer satisfaction. Both of these lead to higher revenue.

The dataset that was used to create my model is the movieLens dataset. Each row in the dataset represents a rating that was submitted. The dataset contains six columns, userId, movieId, rating, timestamp, title, and genres. The userId is a column of unique numbers that each individual user has, to represent who submitted the rating. The movieId is a column of unique numbers that represent what movie the rating was for. The rating column is a number 0-5 which represents how many stars were given as the rating. The timestamp column is a timestamp of when the rating was given. The title is a column that represents the title of the movie that was rated. Finally, the genres column represents what genre the movie is. I used different subsets of this dataset to train my model.

```
head(edx)
```

```
##   userId movieId rating timestamp                          title
## 1      1     122      5 838985046              Boomerang (1992)
## 2      1     185      5 838983525               Net, The (1995)
## 4      1     292      5 838983421               Outbreak (1995)
## 5      1     316      5 838983392               Stargate (1994)
## 6      1     329      5 838983392 Star Trek: Generations (1994)
## 7      1     355      5 838984474       Flintstones, The (1994)
##                            genres
## 1                  Comedy|Romance
## 2           Action|Crime|Thriller
## 4  Action|Drama|Sci-Fi|Thriller
## 5         Action|Adventure|Sci-Fi
## 6 Action|Adventure|Drama|Sci-Fi
## 7         Children|Comedy|Fantasy
```

Method/Analysis

To make my model, before I could start, I had to first both download and understand the data. The edx set is used for training, while the validation set is used for the very final test of the model. The edx dataset has 9,000,055 rows each representing a different rating of a movie by a unique user.

```
dim(edx)
```

```
## [1] 9000055       6
```

Because the dataset is so large, training models using the data would take far too long, so I used the createDataPartition function to further split the edx set into test and training sets. The training set is 90% of the edx set and the test set is 10%.

```
set.seed(1)
sample.kind="Rounding"
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
train_set <- edx[-test_index,]
test_set <- edx[test_index,]
```

Next, I had to do some data cleaning. The first thing that I did was make sure that the userIds and movieIds that are in the test set are also in the training set because otherwise we would have no information about them to use to predict.

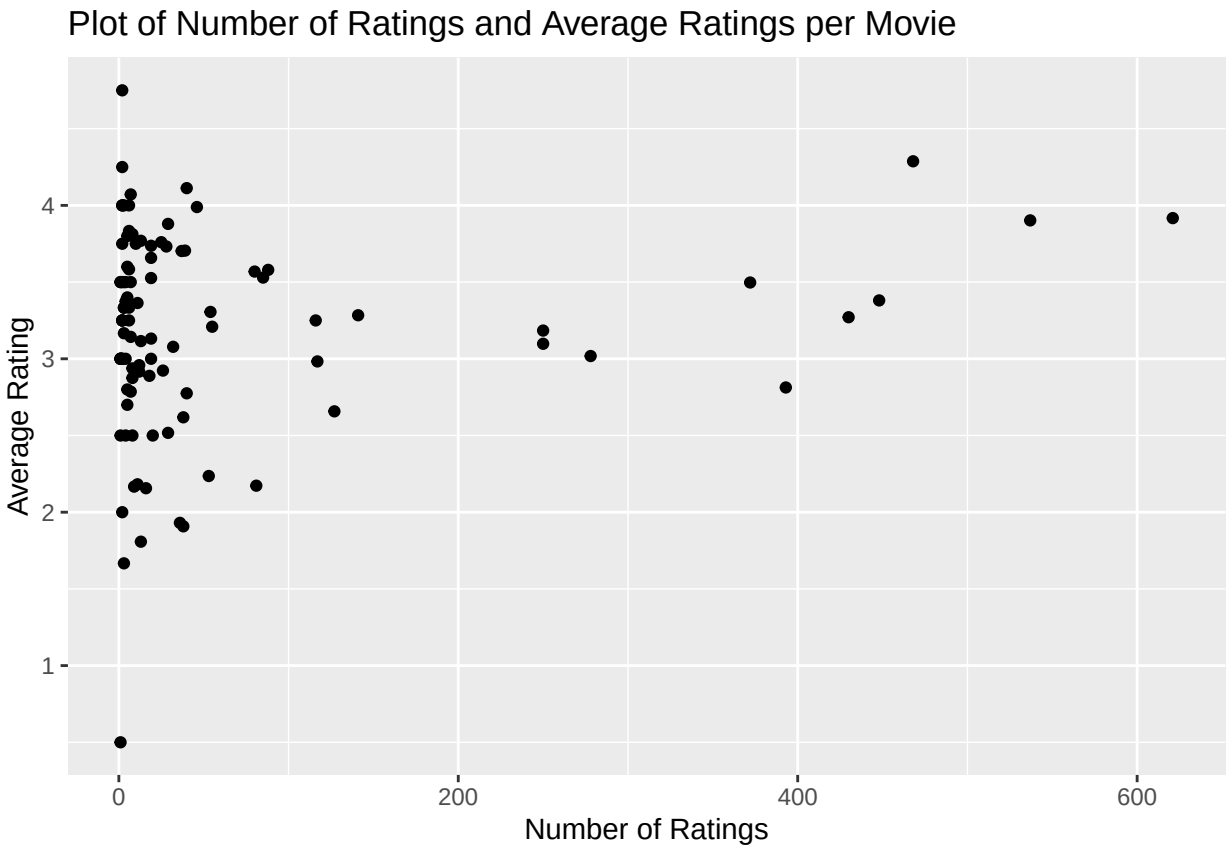
```
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")
```

Second, I had to make sure that there were no NAs in the dataset. An NA is an observation that is blank. This would throw off certain models, so I checked the number of NAs that were in each column, and since this was zero for every column, I didn't have to do anything else regarding NAs.

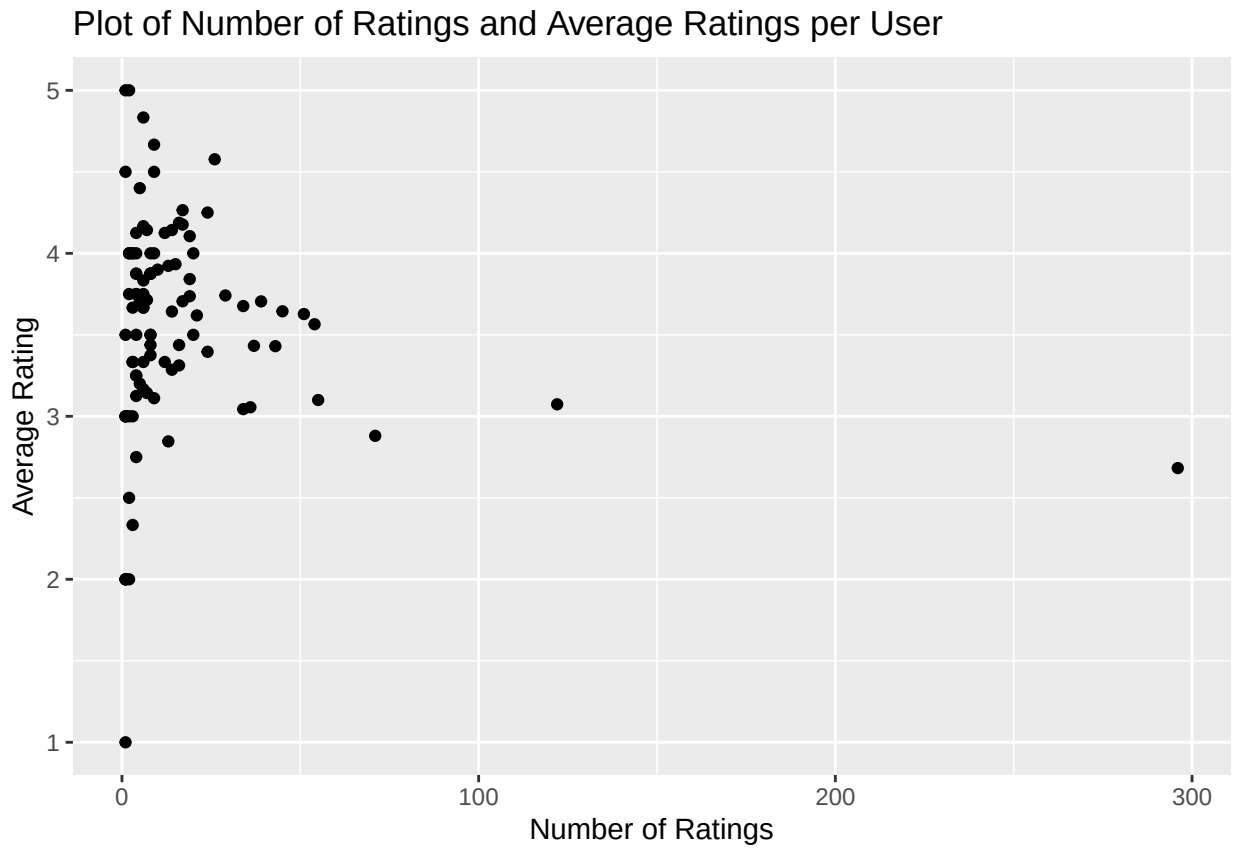
```
sapply(edx,function(x) sum(is.na(x)))
```

```
##    userId   movieId    rating timestamp     title    genres
##         0         0         0         0         0         0
```

The next step was understanding the data so that I could have a better understanding of where to start. I started by looking for distributions of different columns. When plotting the average rating for each movie I noticed that there is a lot of variation of the average depending on the movie. This makes sense because some movies are better than others.



I also decided to do this for users, and similarly, there is a lot of variation showing that different users rate movies more or less strictly.



Throughout the making of this model, I will be using the root-mean-squared-error (RMSE) as a measure of the accuracy of the different models. The formula for the RMSE is:

```
RMSE <- function(actual, pred){
  sqrt(mean((actual-pred)^2))
}
```

The formula gets the average distance of the model's prediction and the actual value, then the square root and square parts of the equation are to get the absolute value of the distance. In this case, an RMSE of 1.0 represents that on average, your models' predictions will be off by one star.

The first method that I used to predict ratings was guessing the average for every single movie. To do this, I simply calculated the average of the train set's ratings. Next, I calculated the RMSE for this prediction using the function that I made. I decided to store my RMSEs in a table so I could easily see what methods worked the best. I got an RMSE of 1.0612018. This is quite inaccurate considering that on average, every rating will be off by more than a whole star. In a five star rating systems such as we have in this data set, this is a significant error.

```
set.seed(1)
sample.kind="Rounding"
#finding the mean of the rating
mu_hat <- mean(train_set$rating)

#Finding RMSE for guessing the mean
mean_rmse <- RMSE(test_set$rating, mu_hat)
```

```
#Created RMSE table that stores the RMSE's for each method
rmse_table <- data.frame(method = "Just the average", RMSE = mean_rmse)
rmse_table
```

```
##              method   RMSE
## 1 Just the average 1.0597
```

After seeing the variability of averages based on the movie, I decided that I needed to factor this into my model. Some movies are generally well liked, while others are not. This movie bias should be taken into account. Therefore, the second method that I used to predict ratings was accounting for movie bias. Movie bias is the assumption that certain movies will on average score higher ratings than others. The equation below represents the model:

$$Y_{u,i} = \mu + b_i + \epsilon_{u,i}$$

Yu,i is the actual rating for each movie user combination, mu is the average rating, bi is the movie bias, and epsilon u,i represents independent errors sampled from the same distribution centered at zero. The hypothesis is that adding in more bias will lower the error. To account for this bias, I averaged the distance from the mean for each movie. Using this average, I created a model and calculated the RMSE of the model, which was 0.9439087, which was an improvement. This improvement follows our hypothesis that adding a variable for the explanation of movie bias will lower the error.

```
set.seed(1)
sample.kind="Rounding"
#Averages of every movie
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu_hat))

#Model that accounts for movie bias
predicted_ratings <- mu_hat + test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_table <- bind_rows(rmse_table, data_frame(method="Movie Effect Model", RMSE = model_1_rmse))
rmse_table
```

```
##               method      RMSE
## 1   Just the average 1.0597002
## 2 Movie Effect Model 0.9430962
```

The third method that I used to predict ratings was accounting for user bias. User bias is the assumption that certain users will on average have higher ratings than others.

$$Y_{u,i} = \mu + b_i + b_u + \epsilon_{u,i}$$

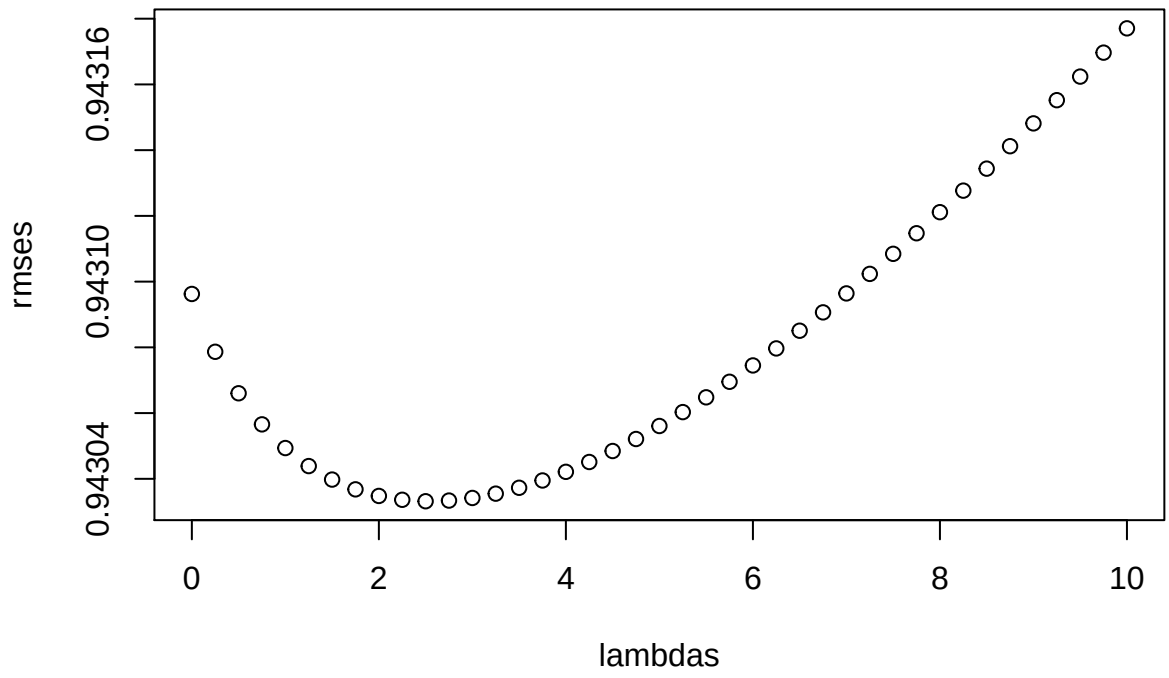
Once again, when looking at the equation, there is a new variable, bu. This is the user bias. Some users generally rate lower than others. Once again adding this new explanation of bias should lower the error. I accounted for the error by averaging the distance between the distance from the average and the average for the movie. Next, I used this average to make a model. Finally, I once again calculated the RMSE, which was 0.8292477, which shows that once again adding more explanation of bias helps lower our error.

```r
set.seed(1)
sample.kind="Rounding"
#Averages of every user
user_avgs <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i))

#Model that accounts for user and movie bias
predicted_ratings2 <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  .$pred

#Adding the next RMSE to the table
model_2_rmse <- RMSE(predicted_ratings2, test_set$rating)
rmse_table <- bind_rows(rmse_table, data_frame(method="Movie + User Effect Model", RMSE = model_2_rmse))
rmse_table
```

```
##                       method      RMSE
## 1          Just the average 1.0597002
## 2        Movie Effect Model 0.9430962
## 3 Movie + User Effect Model 0.8251082
```

The fourth method that I used is called regularization. Regularization is when you shrink the coefficients towards zero, this method is used to control overfitting. The first step of regularization is finding which value of lambda is best. Lambda is a tuning parameter that, and different values of lambda work better for different models. The measurement of how good a value of lambda is is by how low the RMSE is. I found that the best value of lambda was 1.5.

```r
lambdas <- seq(0, 10, 0.25)
mu <- mean(train_set$rating)
just_the_sum <- train_set %>%
  group_by(movieId) %>%
  summarize(s = sum(rating - mu), n_i = n())
rmses <- sapply(lambdas, function(l){
  predicted_ratings <- test_set %>%
    left_join(just_the_sum, by='movieId') %>%
    mutate(b_i = s/(n_i+l)) %>%
    mutate(pred = mu + b_i) %>%
    .$pred
  return(RMSE(predicted_ratings, test_set$rating))
})

#setting lambda to the value that minimizes RMSE
plot(x = lambdas, y = rmses)
```




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

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

Now that I had found the best value of lambda, I regularized both the averages for each movie and the averages for each user. Finally, similarly to what I've done before, I created a model the same way as the last method, by instead with the newly regularized data. The RMSE increased this time, as it was 0.943852. Regularization adds a penalty to users and movies that do not have a large amount of data, so I estimate that these data point were important to the model. This is because there are so many movies that some of them only have a few ratings, however, the model was still able to use these ratings to predict a rating. Because the regularized model penalized these data points, it could not make as accurate of a prediction.

```
#Averages for movies with regularization
movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu_hat)/(n()+lambda), n_i = n())

#Averages for users with regularization
user_reg_avgs <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu_hat - b_i)/(n()+lambda), n_u = n())

#Creating model
```

```
predicted_ratings3 <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  left_join(user_reg_avgs, by='userId') %>%
  mutate(pred = mu_hat + b_i + b_u) %>%
  .$pred

#Adding RMSE to the table
model_3_rmse <- RMSE(predicted_ratings3, test_set$rating)
rmse_table <- bind_rows(rmse_table, data_frame(method="Regularized Movie EFF", RMSE = model_3_rmse))
rmse_table
```

```
##                       method      RMSE
## 1          Just the average 1.0597002
## 2        Movie Effect Model 0.9430962
## 3 Movie + User Effect Model 0.8251082
## 4     Regularized Movie EFF 0.9275464
```

The fifth method that I used was accounting for genre bias. After seeing the improvement that factoring for user and movie bias had on my model, I decided to try another bias. After plotting the average rating for each genre, I realized that there was also a lot of variability depending on the genre. Some genres are more well liked than others. This means that factoring in the genre bias should also improve the model. I factored in this bias the same way that I factored in the other two biases. The RMSE for this model was 0.8285157, another steady improvement.

```
#Method 5: Genre Bias

set.seed(1)
sample.kind="Rounding"
#Averages for each genre
genre_avgs <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu_hat - b_i - b_u))

#Creating a model
predicted_ratings4 <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = mu_hat + b_i + b_u + b_g) %>%
  .$pred

#Adding RMSE to the table
model_4_rmse <- RMSE(predicted_ratings4, test_set$rating)
rmse_table <- bind_rows(rmse_table, data_frame(method="Movie + User + Genre Effect Model", RMSE = model_
rmse_table
```

```
##                              method      RMSE
## 1                  Just the average 1.0597002
## 2                Movie Effect Model 0.9430962
## 3         Movie + User Effect Model 0.8251082
```

```
## 4             Regularized Movie EFF 0.9275464
## 5 Movie + User + Genre Effect Model 0.8243243
```

The sixth method that I used was matrix factorization. An important source of variation is related to the fact that groups of movies and groups of users can have similar rating patterns. This can be observed by studying the residuals and converting our data into a matrix where each user gets a row and each movie gets a column. Matrix Factorization is a machine learning method that creates a matrix or table, which in this case would be with different rows of unique users and different columns of every unique movie. Then, using an algorithm, the method fills in estimates for the rating that each user will give every movie. I used the recosystem package to make the matrix factorization model. The first step was finding the best parameters to use by running a hyperparameter tuning grid search over a supplied range for each parameter. After finding the parameters, I used them to train the algorithm that I used to predict the missing values in the matrix.

```
set.seed(1)
sample.kind="Rounding"
#formating the data
train_format_set <-  with(train_set, data_memory(user_index = userId,
                                                 item_index = movieId,
                                                 rating     = rating))
test_format_set  <-  with(test_set,  data_memory(user_index = userId,
                                                 item_index = movieId,
                                                 rating     = rating))

# Creating the object for the model
r <-  recosystem::Reco()

# Finding the best parameters
par <- r$tune(train_format_set, opts = list(dim = c(10, 20, 30),
                                            lrate = c(0.1, 0.2),
                                            costp_l2 = c(0.01, 0.1),
                                            costq_l2 = c(0.01, 0.1),
                                            nthread  = 4, niter = 10))

# Training the algorithm
r$train(train_format_set, opts = c(par$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9840   1.1086e+07
##    1       0.8754   8.9933e+06
##    2       0.8428   8.3480e+06
##    3       0.8202   7.9566e+06
##    4       0.8037   7.6846e+06
##    5       0.7914   7.4969e+06
##    6       0.7810   7.3512e+06
##    7       0.7722   7.2307e+06
##    8       0.7646   7.1364e+06
##    9       0.7582   7.0555e+06
##   10       0.7526   6.9893e+06
##   11       0.7475   6.9313e+06
##   12       0.7431   6.8827e+06
##   13       0.7391   6.8400e+06
##   14       0.7353   6.7998e+06
```

```
##   15       0.7320   6.7677e+06
##   16       0.7289   6.7355e+06
##   17       0.7261   6.7090e+06
##   18       0.7233   6.6851e+06
##   19       0.7209   6.6590e+06
```

```
#Creating the prediction
predicted_ratings5 <-  r$predict(test_format_set, out_memory())

#Calculating the RMSE
model_5_rmse <- RMSE(predicted_ratings5, test_set$rating)

#Adding it to the table
rmse_table <- bind_rows(rmse_table, data.frame(method = "Matrix Factorization", RMSE = model_5_rmse))
rmse_table
```

```
##                              method      RMSE
## 1                  Just the average 1.0597002
## 2                Movie Effect Model 0.9430962
## 3         Movie + User Effect Model 0.8251082
## 4             Regularized Movie EFF 0.9275464
## 5 Movie + User + Genre Effect Model 0.8243243
## 6              Matrix Factorization 0.7854161
```

Finally, I needed to test my model on the whole validation set. When testing on the test set, matrix factorization was by far the most successful, so I used that for my final model. To create the final model, I used the exact same process as while training on the test set, but I used the larger, validation set.

```
set.seed(1)
sample.kind="Rounding"
#Formatting the edx and validation data set
format_edx <-  with(edx, data_memory(user_index = userId,
                                     item_index = movieId,
                                     rating = rating))
format_validation <-  with(validation, data_memory(user_index = userId,
                                                   item_index = movieId,
                                                   rating = rating))

# Creating the model
r <-  recosystem::Reco()

#Finding the best parameters
par <-  r$tune(format_edx, opts = list(dim = c(10, 20, 30),
                                       lrate = c(0.1, 0.2),
                                       costp_l2 = c(0.01, 0.1),
                                       costq_l2 = c(0.01, 0.1),
                                       nthread  = 4, niter = 10))

# Training the algorithm
r$train(format_edx, opts = c(par$min, nthread = 4, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9728   1.2021e+07
```

```
##    1       0.8726   9.8860e+06
##    2       0.8391   9.1813e+06
##    3       0.8170   8.7577e+06
##    4       0.8014   8.4729e+06
##    5       0.7901   8.2808e+06
##    6       0.7806   8.1322e+06
##    7       0.7724   8.0108e+06
##    8       0.7655   7.9137e+06
##    9       0.7594   7.8313e+06
##   10       0.7541   7.7632e+06
##   11       0.7493   7.7033e+06
##   12       0.7449   7.6514e+06
##   13       0.7411   7.6074e+06
##   14       0.7376   7.5679e+06
##   15       0.7343   7.5327e+06
##   16       0.7313   7.4994e+06
##   17       0.7286   7.4716e+06
##   18       0.7261   7.4479e+06
##   19       0.7237   7.4225e+06
```

```
#Predicting the ratings
predicted_ratings6 <-  r$predict(format_validation, out_memory())

model_6_rmse <- RMSE(predicted_ratings6, validation$rating)
#Adding the RMSE to the table
rmse_table <- bind_rows(rmse_table, data.frame(method = "Matrix Factorization Final", RMSE = model_6_rms
rmse_table
```

```
##                              method      RMSE
## 1                  Just the average 1.0597002
## 2                Movie Effect Model 0.9430962
## 3         Movie + User Effect Model 0.8251082
## 4             Regularized Movie EFF 0.9275464
## 5 Movie + User + Genre Effect Model 0.8243243
## 6               Matrix Factorization 0.7854161
## 7        Matrix Factorization Final 0.7823834
```

Results

This is the entire table of RMSE's that I created for each iteration of my model.

```
rmse_table
```

```
##                              method      RMSE
## 1                  Just the average 1.0597002
## 2                Movie Effect Model 0.9430962
## 3         Movie + User Effect Model 0.8251082
## 4             Regularized Movie EFF 0.9275464
## 5 Movie + User + Genre Effect Model 0.8243243
## 6               Matrix Factorization 0.7854161
## 7        Matrix Factorization Final 0.7823834
```

Going through each RMSE individually. The First RMSE is from guessing the average. The RMSE is 1.0597002, which is very high, however, this makes lots of sense since guessing the average is a fairly simple method. The next RMSE is from the movie effect model. The RMSE for this model is 0.9430962 which is a great improvement. The next RMSE is from the user and movie effect model. The RMSE for this model is 0.8251082, which is another great improvement and is already lower than the goal. The next RMSE is from the regularized model, and the RMSE was 0.9275464, which means that regularization actually made our model worse. This is likely the result of sparse data. Because of this I didn't use it for the final model. The next RMSE is from the genre effect model, and the RMSE is 0.8243243, which is a very small improvement from the previous best model. However, this makes sense because the genre effect is somewhat accounted for by the movie effect since we have already accounted for each movies average which genre is one part of. The next model is the matrix factorization model. The RMSE is 0.7860644, which is a great improvement from the last one, which is very impressive considering the previous model was already far under the goal. Finally, I tested the matrix factorization on the whole dataset for my final model, and the RMSE was 0.7829283.

Conclusion

I started with downloading, cleaning, and examining the data. Once I was ready to make models, I started by using the simplest model that I could think of which was the average rating. This proved to be a very poor predictor with above a 1.0 RMSE. The next few models were several iterations of models that each factored in a new bias, including movie, user, and genre bias. These got progressively less effective because the bias was already removed by each previous model. The next model used regularization, however, it made the model worse. The final test model used matrix factorization to make predictions and it was the best model. Finally, we tested the final model on the whole dataset.

This model performed well, but it still has some limitations. First, the model takes quite a long time to run. This is important because the dataset changes every time a new rating occurs, so the model would need to be rerun periodically. The final matrix factorization took over an hour but could be faster on a computer with more processors and memory. Another limitation is that the model relies solely on ratings, however, lots of users don't rate movies.

A way to combat the reliance on ratings is by adding another piece of data, such as time watched. That means that even if a user did not rate a movie, a model could still learn from the user's choice because if they stopped watching very early into a movie and never came back, the model could learn that the user did not like that movie.

Overall, I thoroughly enjoyed this project as it gave me an opportunity to learn more about machine learning and prediction models. Before this projects, I was not sure how to do matrix factorization. Over the course of the project, I learned a lot about how to do it and how it works.