2nd November, 2022

CAPTSONE PROJECT

MOVIE RATING PREDICTION ALGORITHM

KEVIN OCANSEY

Submitted to Edx Course PH125.9x

ABSTRACT

My submission for the Capstone project (Movie Recommendation system) of the Edx Data Science course. The initial code below was made available by the facilitators of the course. I was tasked to build an algorithm that predicts movie ratings using movielens data set as the training set. I am to improve on the models from the previous course (Machine learning) in order to reduce the RMSE to a value below 0.86. I had an amazing time taking this program. A special thanks to Rafael Irizarry and everyone who made this possible

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

# Note: this process could take a couple of minutes

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(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")

library(tidyverse)
library(caret)
library(data.table)

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- tempfile()
download.file("https://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))


```

```r
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)`
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)

```

OVERVIEW

```
1.  head(edx)
2.  dim(edx)
3.  edx %>% summarize(n_users = n_distinct(userId),
4.            n_movies = n_distinct(movieId))
5.
```

The edx dataset has:

- 9000055 rows and 6 columns
- 69878 unique users
- 10677 unique movies

After skimming through the dataset, I realized there was a particular movie with movieId = 8606 titled 'Pull my Daisy'. It was the only movie with no genre listed hence, I decided to remove it from the project. The decision was solely due to the fact that it was the only movie with no genre.

This reduces our total rows to 9000048 and our unique movies to 10676.

RMSE

Residual mean squared error, is the difference between the actual value and what our model predicts. Throughout the report we will use this function to determine the error between our predicted values and the actual values in our edx dataset for every model we develop**.** Keep in mind, the smaller the error the more accurate our model is. The formula looks like this:

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

$\hat{y}_{u,i}$ represents the predicted value of movie rating. Rating by user *u* and rating of the movie *i*

$y_{u,i}$ represents the actual value of movie rating. Rating by user *u* and rating of the movie with movie id *i.*

N is the total number of user/movie combinations

For our (in this case my) porject if the RMSE is larger than 1 it means our typical error is greater than a star rating. The main aim is to reduce the error below 0.865

VISUALIZATION

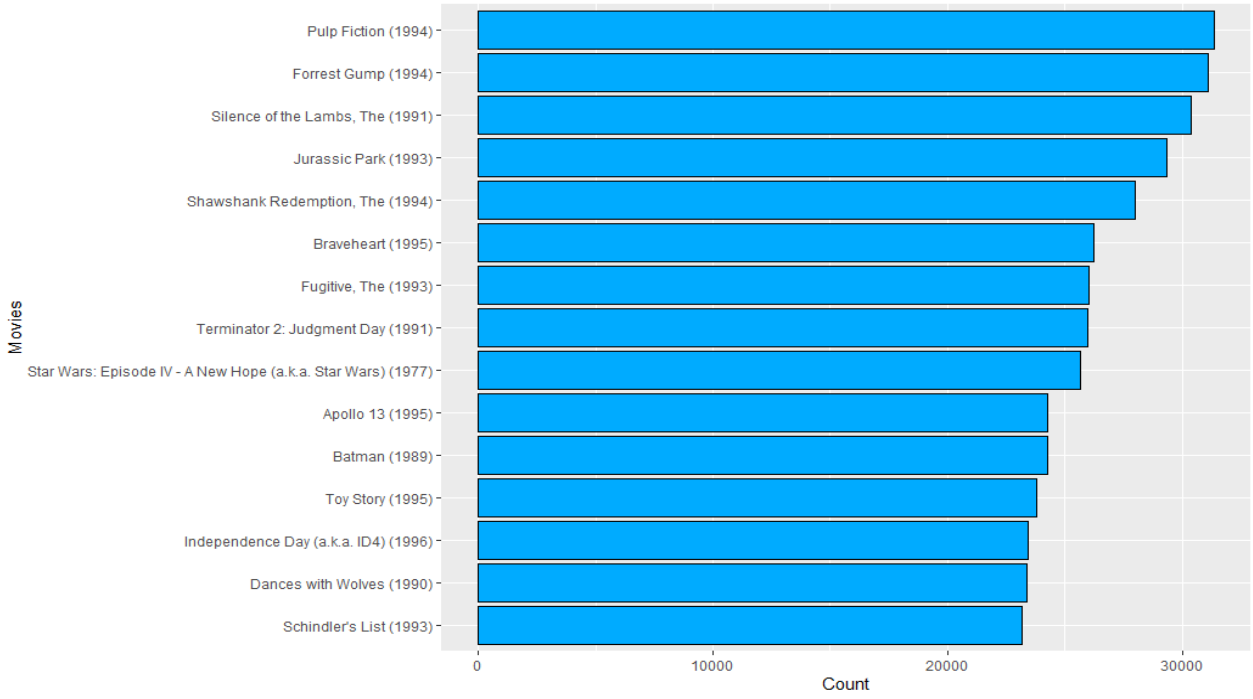

Fig a

Most rated movies

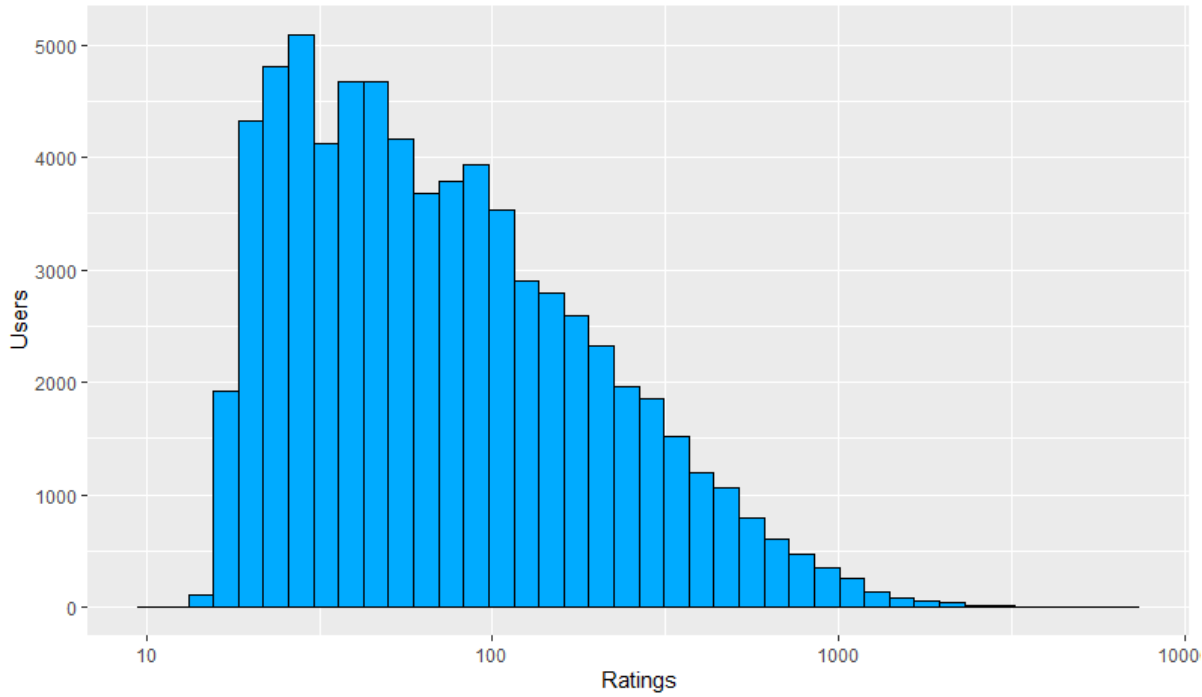

Fig b

Ratings per Users

MACHINE LEARNING

To finally get an RMSE lower than 0.86, I will slowly improve on a model from the Machine Learning course. It starts with a simple model where we predict the same rating for all movies. Next we will later consider the movie bias and user bias/effect, which explains variability in ratings.

Lastly, I will improve the model with Regularization and Matrix factorization.

1. Simple model

Keep in mind our next 3 models will extend from this first model. I will be referencing this model and a couple others from the data science book (Introduction to Data Science by *Rafael A. Irizarry).*

For our first model we calculate the mean of our ratings in the training set and use that figure, as prediction for all movies in our test set.

Our simple model looks like this mathematically ($Y_{u,i} = \mu + \varepsilon_{u,i}$), where 'Y' represents our predicted rating, 'u' represents our average rating for all movies and epsilon represents any minor error

Calculating the RMSE, we get 1.06. our goal is a rmse below 0.86

```r
#1. Simple model
mu_ratings <- mean(train_set$rating)
mu_ratings

simple_model_rmse <- RMSE(test_set$rating, mu_ratings)

#  Meanwhile i would create a table to keep track of all the rmse results as we improve the model
rmse_results <- tibble(method = "Simple Model using Average", RMSE = simple_model_rmse)

```

The rmse_results table for now, looks like this:

Model	Rmse
Simple Model with mean	1.06

2. Simple Model + Movie effect

With our second model we focus on movie bias. Obviously, certain movies will be rated differently than others. These includes blockbuster movies, movie franchise, movies with highly rated directors and others with a specific cast and or highly rated actors.

We can add the term movie bias b_i to improve our previous model to get: $Y_{u,i} = \mu + b_i + \varepsilon_{u,i}$

```
1.  # 2. Movie effect model + simple model.  to measure their bias from the avg movie
    ratings we got from mu_ratings
2.  mu <- mean(train_set$rating)
3.  movie_avgs <- train_set %>%
4.    group_by(movieId) %>%
5.    summarize(b_i = mean(rating - mu))
6.
7.
8.  movie_avgs %>% ggplot(aes(b_i)) +
9.    geom_histogram(color = "black", fill = "#00abff", bins = 10) +
10.   xlab("Movie Bias") +
11.   ylab("Count")
12.
13.
14. # Now adding b_i into our first simple model gives us something like this  yu,i = μ_hat
    + b_hati
15. predicted_ratings <- mu + test_set %>%
16.   left_join(movie_avgs, by='movieId') %>%
17.   pull(b_i)
18. movie_effect <- RMSE(predicted_ratings, test_set$rating)
19.
20.
21. rmse_results <- add_row(.data = rmse_results, method= 'Simple Model + Movie effects',
    RMSE =  movie_effect )
22.
```

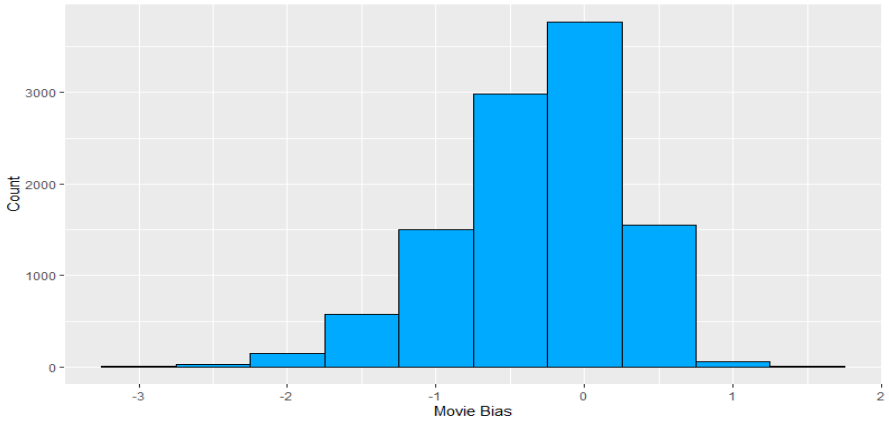

Fig c

Variability in movie rating

The graph above shows the variability in movie ratings. This is why the movie effect b_i is very important. It attempts to counter variability in the equation.

The RMSE results table:

Model	Rmse
Simple Model with mean	1.06
Simple Model + Movie Bias/Effect	0.944

3. Simple Model + Movie effect + User effect

Following our last model, we augment the user's effect b_u into our previous model (Simple model + movie effect).

```
# 3.User Effect

train_set %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating)) %>%
  filter(n()>=100) %>%
  ggplot(aes(b_u)) +
  geom_histogram(bins = 30, fill = "#00abff")


# User Effect Model
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)
user_effect <- RMSE(predicted_ratings, test_set$rating)


rmse_results <- add_row(.data = rmse_results, method= 'Simple Model + Movie Effect +
  User effects', RMSE = user_effect )

# Now We  need to get an Rmse lower than 0.86

```

Taking a look at the graph below we notice variability in users rating. Again, just like the movie effect b_i, the user effect b_u, will help us counter variability, by providing a closer prediction for movie ratings even when users give bad ratings for a movie with generally good ratings or vice versa.

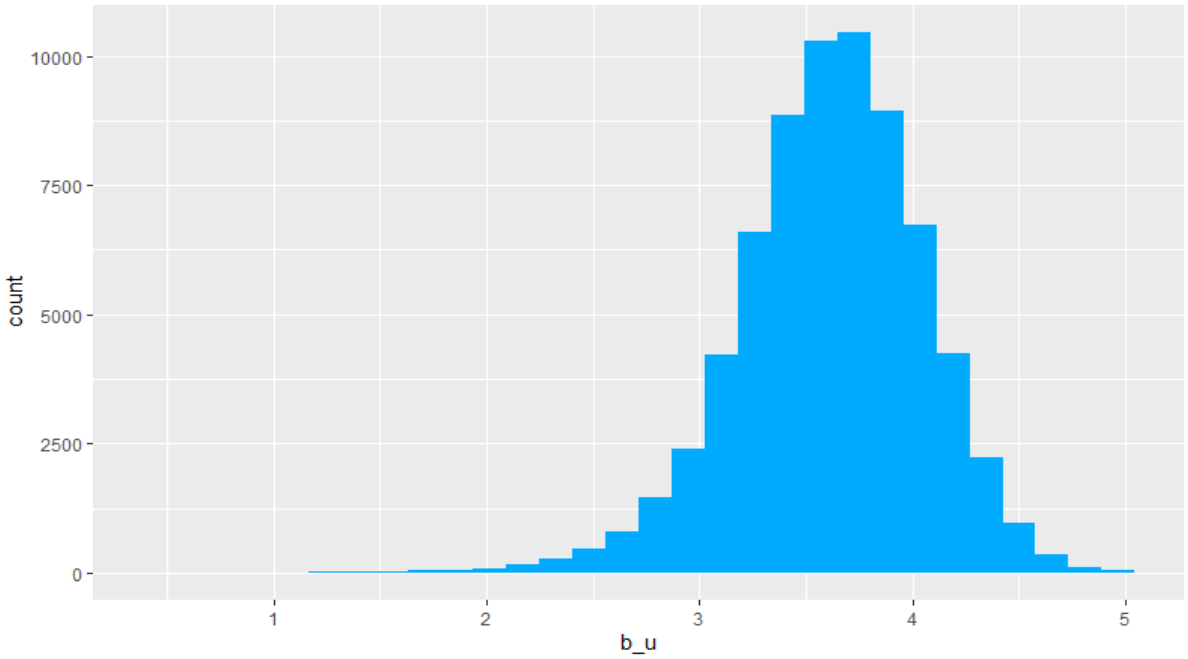

Fig d

Variability in User Rating

The RMSE results table:

Model	Rmse
Simple Model with mean	1.06
Simple Model + Movie Bias/Effect	0.944
Simple Model + Movie Bias/Effect + User effect/bias	0.865

4. Regularization

There are certain movies that are rated by only a few users. For example, if a movie was rated by only two users and had an average rating of 4.5. It would be unadvisable to recommend that movie to another user because there is uncertainty about its true rating. Uncertainty, results in wrong predictions, and large errors, hence an increase our RMSE.

The main idea behind regularization is to constrain the total variability of the effects sizes. We can use regularization, to penalize large estimates formed by small sample sizes hence reducing their effect. The right penalty to use, lambda, can be derived using cross validation. Check the code below.

```
1.  lambdas <- seq(0, 10, 0.25)
2.  rmses <- sapply(lambdas, function(x){
3.    b_i <- train_set %>%
4.      group_by(movieId) %>%
5.      summarize(b_i = sum(rating - mu)/(n()+x))
6.    b_u <- train_set %>%
7.      left_join(b_i, by = "movieId") %>%
8.      group_by(userId) %>%
9.      summarize(b_u = sum(rating - b_i - mu)/(n()+x))
10.   predicted_ratings <- test_set %>%
11.     left_join(b_i, by = "movieId") %>%
12.     left_join(b_u, by = "userId") %>%
13.     mutate(pred = mu + b_i + b_u) %>%
14.     pull(pred)
15.   return(RMSE(predicted_ratings, test_set$rating))
16. })
17.
18. lambda_rmse <-  lambdas[which.min(rmses)]
19. qplot(lambdas, rmses)
20. # lambda of 4.75 provides the lowest RMSE error
21.
22. rmse_results <- add_row(.data = rmse_results, method= 'Simple Model + Movie Effect +
    User effects + Regularization', RMSE = min(rmses))
23.
```

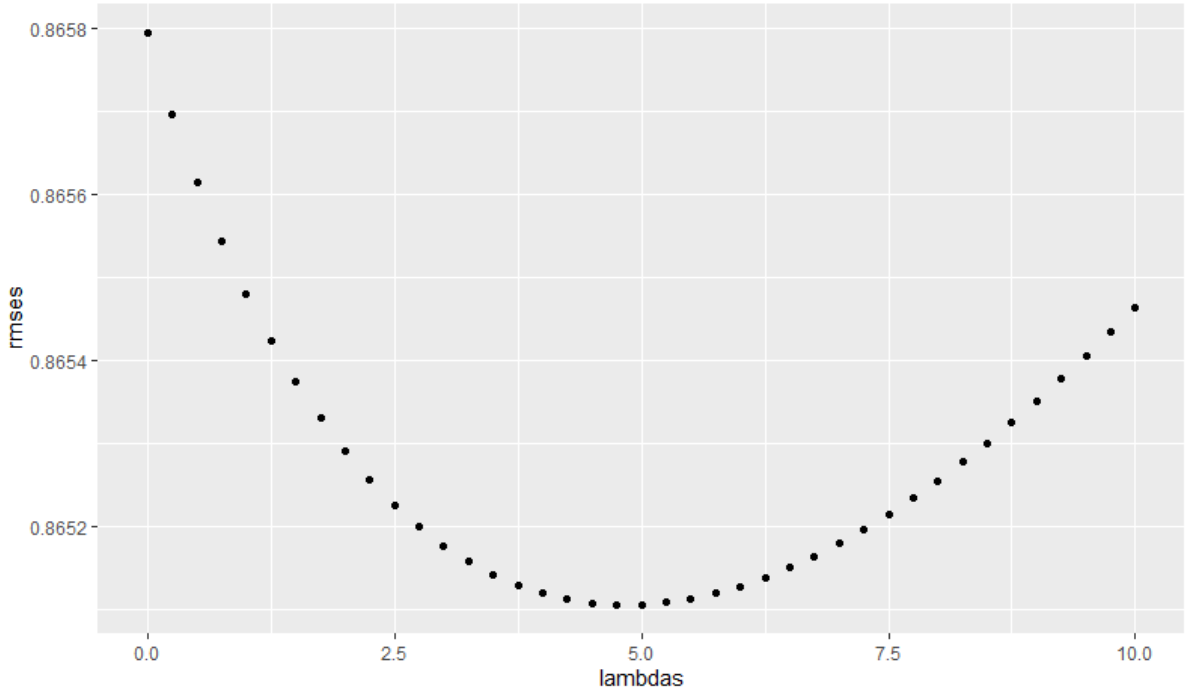

Fig e

Finding the optimal penalty (lambda)

After inserting our penalty 0.475, I updated the rmse table

The RMSE results table:

Model	Rmse
Simple Model with mean	1.06
Simple Model + Movie Bias/Effect	0.944
Simple Model + Movie Bias/Effect + User effect/bias	0.865
Regularized Movie + User effect/bias	0.8651

Regularization only made a tiny difference, however, it was still essential

5. Factorization using Recosystem

Factorization is an essential technique used by most recommendation systems. It is done by converting data sets into matrices and finding similar patterns between one matrix and the other. The first matrix is usually the users and the other is usually the predictive factors/attributes.

Searching for other ways to factorize the edx data set I came across 'recosystem' library from a solved question on Rpubs.

Overview of Recosystem

6. You create a model object using Reco()
7. With that same object you call the tune() method to select the best tuning parameters. So for example I saved the Reco() object model into r and called the method by r$tune()
8. You can train the model by calling the $train() method.
9. After you're done training you can call the $predict() method on the test set. After you ran the r$train, it is saved in memory so you can run your predict method straight away on your test set just using the same model object. So r$predict(test_set, out_memory())

I have provided a link below in case anyone reading this would love to understand it in depth.

https://www.r-bloggers.com/2016/07/recosystem-recommender-system-using-parallel-matrix-factorization/

```
1.  library(recosystem)
2.  set.seed(1, sample.kind="Rounding")
3.  train_reco <- with(train_set, data_memory(user_index = userId, item_index = movieId,
    rating = rating))
4.  test_reco <- with(test_set, data_memory(user_index = userId, item_index = movieId,
    rating = rating))
5.  r <- Reco()
6.
7.  tuned_reco <- r$tune(train_reco, opts = list(dim = c(20, 30),
8.                                               costp_l2 = c(0.01, 0.1),
9.                                               costq_l2 = c(0.01, 0.1),
10.                                              lrate = c(0.01, 0.1),
11.                                              nthread = 4,
12.                                              niter = 10))
13.
14. r$train(train_reco, opts = c(tuned_reco $min, nthread = 4, niter = 30))
15. results_reco <- r$predict(test_reco, out_memory())
16.
17.
18. factorization_rmse <- RMSE(results_reco, test_set$rating)
19.
20. rmse_results <- add_row(.data = rmse_results, method= 'Factorization using Reco
    library', RMSE = factorization_rmse)
21. # our Rmse now is below 0.86 at 0.79
22.
```

The RMSE results table:

Model	Rmse
Simple Model with mean	1.06
Simple Model + Movie Bias/Effect	0.944
Simple Model + Movie Bias/Effect + User effect/bias	0.865
Regularized Movie + User effect/bias	0.8651
Factorization with Recosystems	0.789

6. Final Validation

With the validation dataset I tested the Recosystem model and had an RMSE below 0.86 at 0.780.

```
# Validation with our VALIDATION data set, with edx as our training data set
set.seed(1, sample.kind="Rounding")
edx_reco <- with(edx, data_memory(user_index = userId, item_index = movieId, rating = rating))
validation_reco <- with(validation, data_memory(user_index = userId, item_index = movieId, rating = rating))
r <- Reco()

tuned_reco <- r$tune(edx_reco, opts = list(dim = c(20, 30),
                                           costp_l2 = c(0.01, 0.1),
                                           costq_l2 = c(0.01, 0.1),
                                           lrate = c(0.01, 0.1),
                                           nthread = 4,
                                           niter = 10))

r$train(edx_reco, opts = c(tuned_reco$min, nthread = 4, niter = 30))

final_reco <- r$predict(validation_reco, out_memory())
final_rmse <-  RMSE(final_reco,validation$rating)
rmse_results <- add_row(.data = rmse_results, method= 'Final Rmse', RMSE = final_rmse)

```

Final RMSE results table:

Model	Rmse
Simple Model with mean	1.06
Simple Model + Movie Bias/Effect	0.944
Simple Model + Movie Bias/Effect + User effect/bias	0.865
Regularized Movie + User effect/bias	0.8651
Factorization with Recosystems	0.789
Final Validation with validation dataset	0.780