movielens

Maris Sekar

08/11/2020

Executive Summary

A movie recommendation system was built as part of this project using five machine learning models and a baseline average model. The dataset "MovieLens" comes from the movielens.org research site run by GroupLens Research at the University of Minnesota where movie ratings are collected form users. The dataset provides ten million movie ratings for about 10,000 movies rated by about 70,000 users.90% of the dataset was used as training data and 10% was used as the validation dataset.

The goal of the project was to determine a machine learning model that has a root mean square error (RMSE) of less than 0.86490 with the validation dataset. We will see that the Matrix Factorization model has the lowest RMSE of 0.7836. What this means is that this model can predict the movie ratings given the user and movie to a high degree of accuracy.

The steps involved in the search for the optimal include performing exploratory data analysis to understand the data, followed by looking at the average rating, movie effect, combined movie and user effect, the regularized movie effect, the regularized movie and user effect and finally Matrix Factorization. We will see that the regularized movie and user effect RMSE is already lower than the goal of 0.86490 but given that the Matrix Factorization model was much effective with a much lower RMSE it was hard not to continue with the implementation. The Reco R package was used for the Matrix Factorization method.

Methods/Analysis

Lets perform some exploratory data analysis on the training dataset:

Lets look at the dimensions of the training data

```
dim(edx)
```

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

Lets look at the highest rated movies:

```
edx %>% group_by(movieId, title) %>% summarize(count=n()) %>% arrange(desc(count))
```

```
## `summarise()` regrouping output by 'movieId' (override with `.groups` argument)
```

```
## # A tibble: 10,677 x 3
## # Groups:   movieId [10,677]
##    movieId title                                                            count
##      <dbl> <chr>                                                            <int>
## 1      296 Pulp Fiction (1994)                                     31362
## 2      356 Forrest Gump (1994)                                     31079
## 3      593 Silence of the Lambs, The (1991)                        30382
## 4      480 Jurassic Park (1993)                                    29360
## 5      318 Shawshank Redemption, The (1994)                        28015
## 6      110 Braveheart (1995)                                       26212
## 7      457 Fugitive, The (1993)                                    25998
## 8      589 Terminator 2: Judgment Day (1991)                       25984
## 9      260 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977) 25672
## 10     150 Apollo 13 (1995)                                        24284
## # ... with 10,667 more rows
```

We can see that "Pulp Fiction, Forrest Gump and The Silence of the Lambs" are the movies that are most rated.

Lets look at the users that rate frequently:

```
edx %>% group_by(userId) %>% summarize(count=n()) %>% arrange(desc(count))
```

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

```
## # A tibble: 69,878 x 2
##    userId count
##     <int> <int>
##  1  59269  6616
##  2  67385  6360
##  3  14463  4648
##  4  68259  4036
##  5  27468  4023
##  6  19635  3771
##  7   3817  3733
##  8  63134  3371
##  9  58357  3361
## 10  27584  3142
## # ... with 69,868 more rows
```

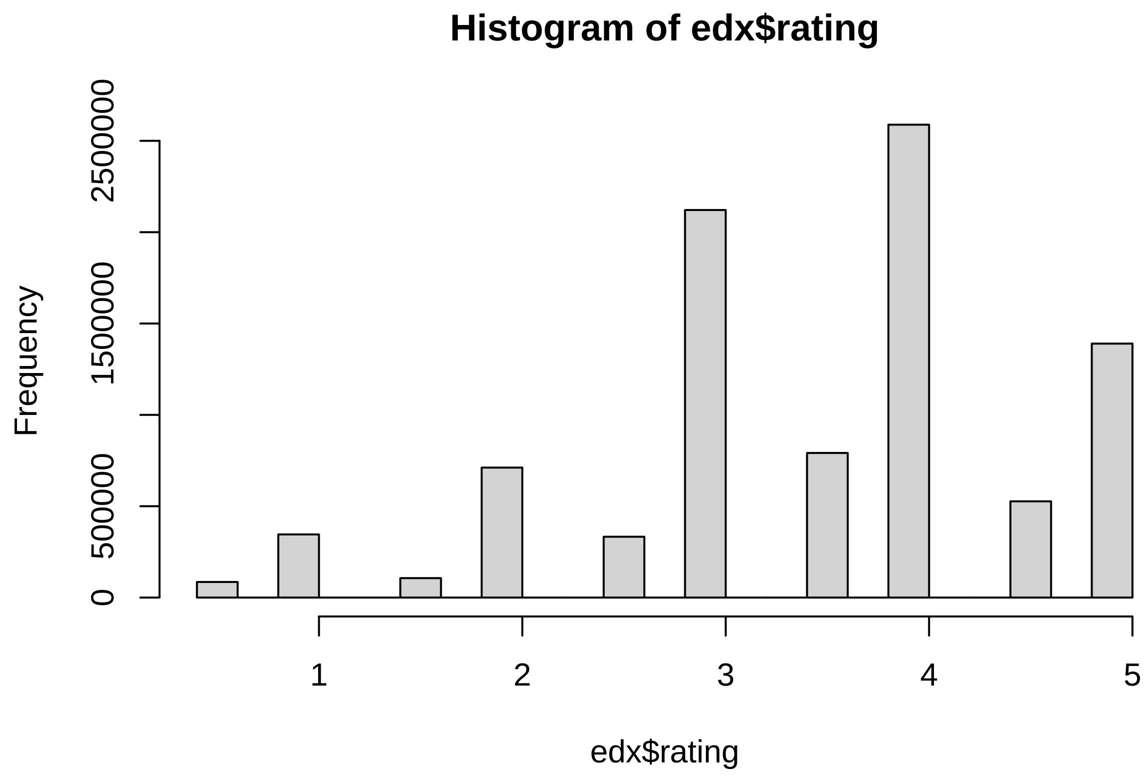
The top 2 users have given more than 6000 ratings followed by around 4500 for the next user.

```
edx %>% select(rating) %>% group_by(rating) %>% summarize(count=n()) %>% arrange(desc(count))
```

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

```
## # A tibble: 10 x 2
##    rating   count
##     <dbl>   <int>
##  1    4   2588430
##  2    3   2121240
##  3    5   1390114
##  4    3.5  791624
##  5    2    711422
##  6    4.5  526736
##  7    1    345679
##  8    2.5  333010
##  9    1.5  106426
## 10    0.5   85374
```

```r
# Lets look at the distribution for ratings
hist(edx$rating)
```



The rating four was given the most, followed by three and then five. The half ratings were the least popular amount users. This shows that users generally gave positive ratings in general and avoided half ratings.

The first model we are going to use is the averaging model. This model takes the average rating and uses that as the prediction across all expected values for the validation dataset.

```r
# Get the average, use it for prediction and record RMSE
mu_hat <- mean(edx$rating)
rmse_avg <- RMSE(edx$rating, mu_hat)
rmse_results <- tibble(method = "Just the average", RMSE = rmse_avg)
```

The second model obtains the movie effect by grouping the movies and summarizing by the movie's average rating after subtracting the mean of the train dataset. We subtract the mean of the whole dataset to bring the values closer and minimize the spread of data.

```r
# Assign edx as the train_set table and validation as the test_set table
train_set <- edx
test_set <- validation

# Calculate the average
mu <- mean(train_set$rating)

# Calculate the movie effect by looking at the mean when grouped by movie
```

```
movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - mu))
```

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

```
# Join with validation test to compare with movies that have been predicted
predicted_ratings <- mu+test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  .$b_i

# Calculate RMSE and record
model_1_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie Effect Model",
                                     RMSE = model_1_rmse ))
```

```
## Warning: `data_frame()` is deprecated as of tibble 1.1.0.
## Please use `tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```

We can see that this model is a little better than the average model.

Now lets see how a model that takes into account user effect along with movie effect performs. The movie effect b_i is combined with the user effect b_u. The user effect b_u is calculated similar to the movie effect above. These are then added to the mean to form the prediction of this model.

```
# Add user effect to the movie effect and get RMSE for comparison
# with validation test
user_avgs <- test_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)
```

```
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred = mu + b_i + b_u) %>%
  .$pred
model_2_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + User Effects Model",
                                     RMSE = model_2_rmse ))
```

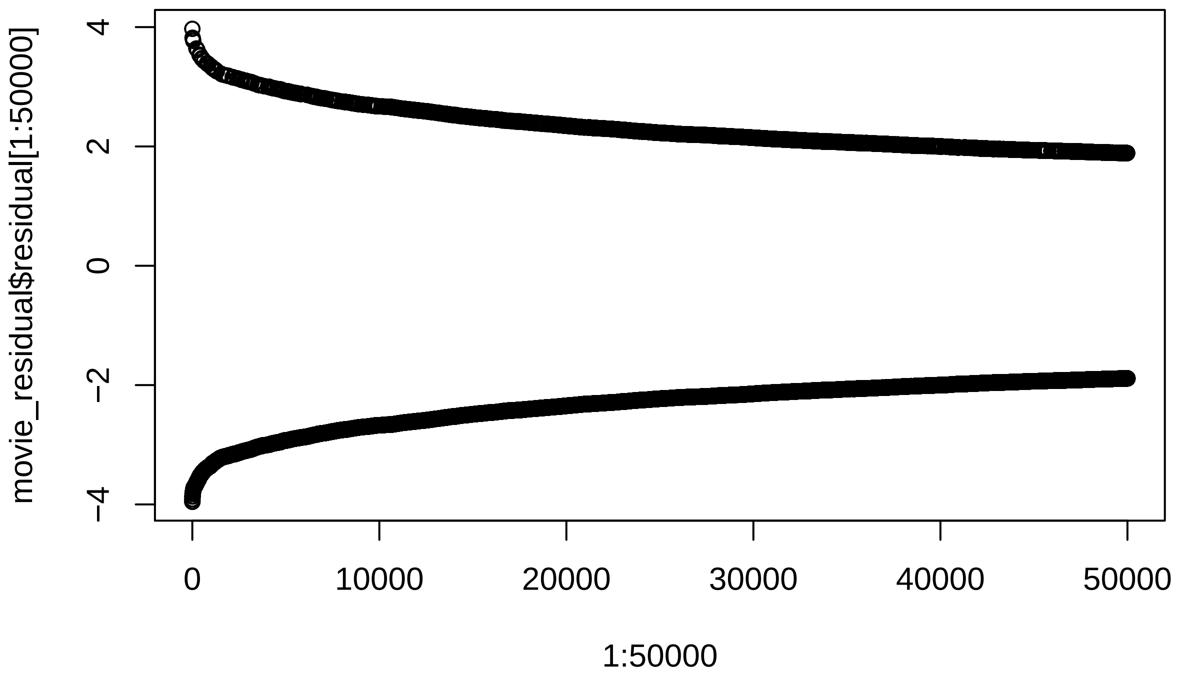
This model is better than the movie effect alone.

Next we will look at the residual for the movie effect to see if regularization will help for the data set.

```
# Calculate residual and plot it for the movie effect.
test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(residual = rating - (mu + b_i)) %>%
  arrange(desc(abs(residual))) %>%
  select(title,  residual) %>% slice(1:10) %>% knitr::kable()
```

title	residual
Pokémon Heroes (2003)	3.970803
Shawshank Redemption, The (1994)	-3.955131
Shawshank Redemption, The (1994)	-3.955131
Shawshank Redemption, The (1994)	-3.955131
Godfather, The (1972)	-3.915366
Godfather, The (1972)	-3.915366
Godfather, The (1972)	-3.915366
Usual Suspects, The (1995)	-3.865854
Usual Suspects, The (1995)	-3.865854
Usual Suspects, The (1995)	-3.865854

```
movie_residual <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(residual = rating - (mu + b_i)) %>%
  arrange(desc(abs(residual))) %>%
  select(title,  residual)
plot(1:50000, movie_residual$residual[1:50000])
```



From the above plot it can be seen that the residual is symmetric around 0.

Lets look at the top 10 movies with the most positive movie effect.

```
# Get distinct movie titles
movie_titles <- edx %>%
  select(movieId, title) %>%
  distinct()

movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i
Hellhounds on My Trail (1999)	1.487535
Satan's Tango (Sátántangó) (1994)	1.487535
Shadows of Forgotten Ancestors (1964)	1.487535
Fighting Elegy (Kenka erejii) (1966)	1.487535
Sun Alley (Sonnenallee) (1999)	1.487535
Blue Light, The (Das Blaue Licht) (1932)	1.487535
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237535
Human Condition II, The (Ningen no joken II) (1959)	1.237535
Human Condition III, The (Ningen no joken III) (1961)	1.237535
Constantine's Sword (2007)	1.237535

Now lets look at the 10 movies with the least negative movie effect.

```r
movie_avgs %>% left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i) %>%
  slice(1:10) %>%
  knitr::kable()
```

title	b_i
Besotted (2001)	-3.012465
Hi-Line, The (1999)	-3.012465
Accused (Anklaget) (2005)	-3.012465
Confessions of a Superhero (2007)	-3.012465
War of the Worlds 2: The Next Wave (2008)	-3.012465
SuperBabies: Baby Geniuses 2 (2004)	-2.717822
Hip Hop Witch, Da (2000)	-2.691037
Disaster Movie (2008)	-2.653090
From Justin to Kelly (2003)	-2.610455
Criminals (1996)	-2.512465

We can see there is a more negative movie effect than positive effect

Next, inspect the top 10 movies with the most positive effect along with the number of reviews.

```r
train_set %>% dplyr::count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

```
## Joining, by = "movieId"
```

title	b_i	n
Hellhounds on My Trail (1999)	1.487535	1
Satan's Tango (Sátántangó) (1994)	1.487535	2
Shadows of Forgotten Ancestors (1964)	1.487535	1
Fighting Elegy (Kenka erejii) (1966)	1.487535	1
Sun Alley (Sonnenallee) (1999)	1.487535	1
Blue Light, The (Das Blaue Licht) (1932)	1.487535	1
Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)	1.237535	4
Human Condition II, The (Ningen no joken II) (1959)	1.237535	4
Human Condition III, The (Ningen no joken III) (1961)	1.237535	4
Constantine's Sword (2007)	1.237535	2

We can see that the movies don't have many reviews.

Inspect the least 10 movies with the most negative effect and the number of reviews.

```r
train_set %>% dplyr::count(movieId) %>%
  left_join(movie_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

```
## Joining, by = "movieId"
```

title	b_i	n
Besotted (2001)	-3.012465	2
Hi-Line, The (1999)	-3.012465	1
Accused (Anklaget) (2005)	-3.012465	1
Confessions of a Superhero (2007)	-3.012465	1
War of the Worlds 2: The Next Wave (2008)	-3.012465	2
SuperBabies: Baby Geniuses 2 (2004)	-2.717822	56
Hip Hop Witch, Da (2000)	-2.691037	14
Disaster Movie (2008)	-2.653090	32
From Justin to Kelly (2003)	-2.610455	199
Criminals (1996)	-2.512465	2

We can see that the movies dont appear to be top 10 rated and least 10 rated movies in reality. This could be because people tend to review popular reviews and this is not captured in the top 10.

This warrants regularization to be used. We want to first check the regularization rate histogram to get an idea of the how the regularization affects the movie effect. Lets set a regularization rate that gives a normal distribution to b_i and performs regularization - pulls values closer to 0.

Lets first calculate the residual with regularization rate of 0.

```r
lambda <- 0
mu <- mean(train_set$rating)
movie_reg_avgs_no <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n()) %>%
  mutate(eff="No Regularization")
```

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

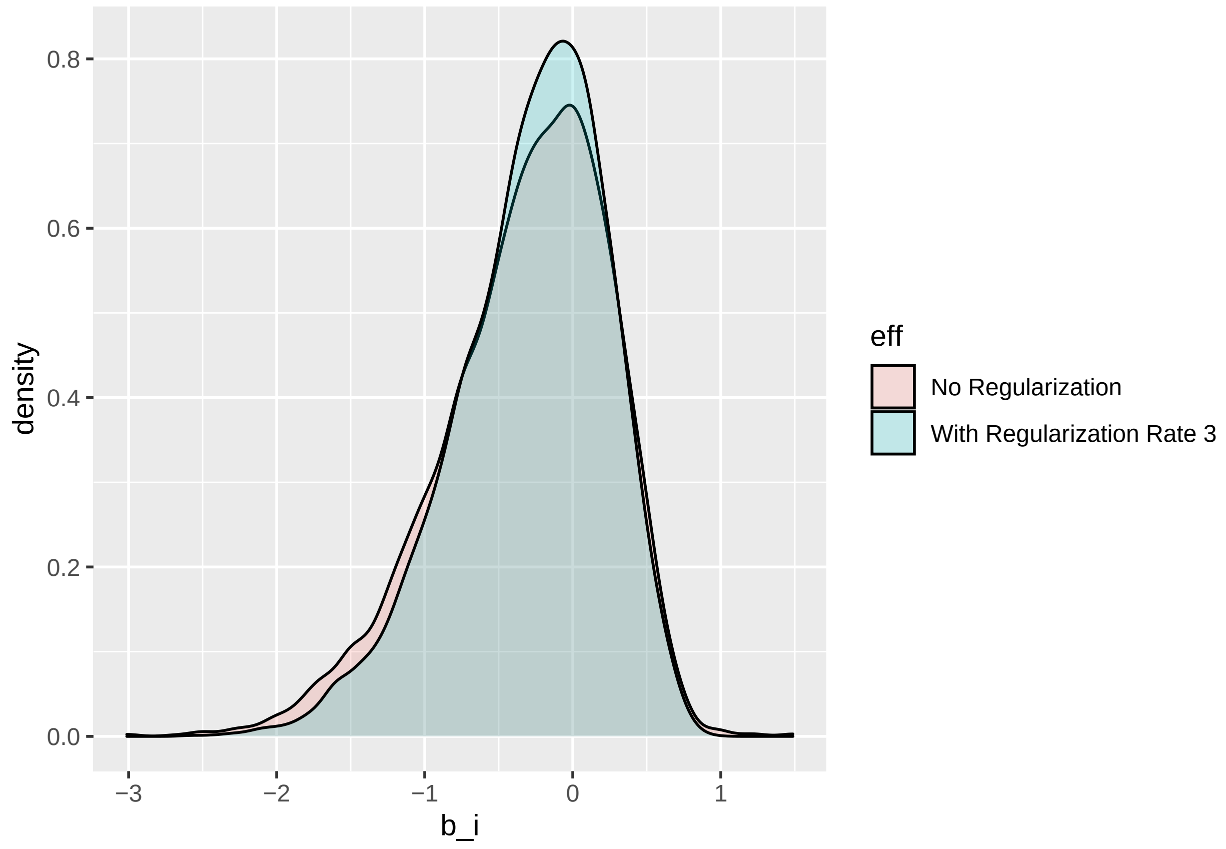
Next, lets calculate the residual with a regularization rate of 3.

```r
lambda2 <- 3
movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda2), n_i = n()) %>%
  mutate(eff="With Regularization Rate 3")
```

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

Lets plot the distribution for the regularized movie effect and non-regularized movie effect.

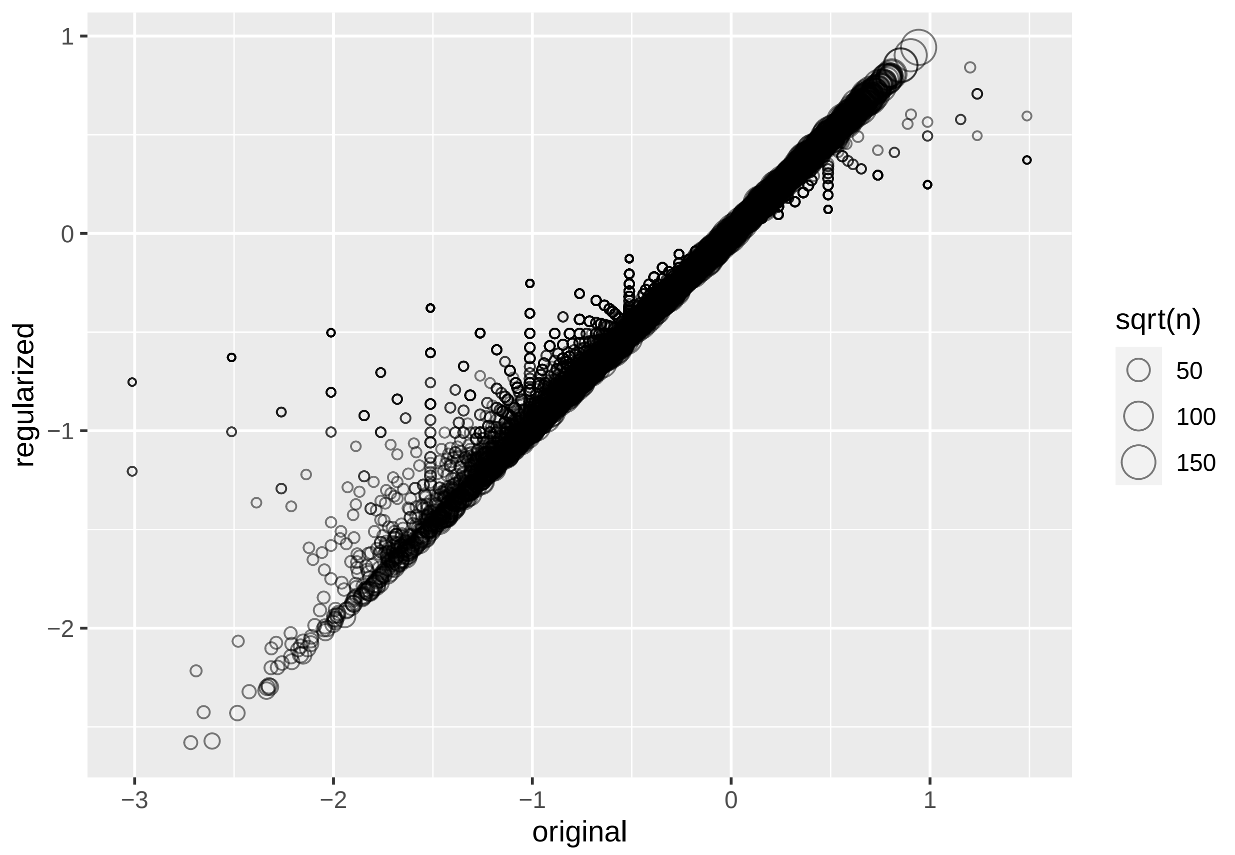
```
reg_movie_eff <- rbind(movie_reg_avgs_no, movie_reg_avgs)
ggplot(reg_movie_eff, aes(b_i, fill = eff)) + geom_density(alpha = 0.2)
```



We can see how the regularization rate helps pull the movie effect closer to zero.

Lets compare this effect using a scatter plot.

```
# Plot the regularized vs. original movie effect
data_frame(original = movie_avgs$b_i,
           regularized = movie_reg_avgs$b_i,
           n = movie_reg_avgs$n_i) %>%
  ggplot(aes(original, regularized, size=sqrt(n))) +
  geom_point(shape=1, alpha=0.5)
```



We can see that the regularization helps pull the movie effect closer to 0 as well as how the movies with more reviews are unaffected by the effect which is as expected.

Now, lets inspect the top 10 movies with positive movie effect again.

```
train_set %>%
  dplyr::count(movieId) %>%
  left_join(movie_reg_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(desc(b_i)) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

```
## Joining, by = "movieId"
```

title	b_i	n
Shawshank Redemption, The (1994)	0.9425650	28015
Godfather, The (1972)	0.9027482	17747
Usual Suspects, The (1995)	0.8532702	21648
Schindler's List (1993)	0.8509180	23193
More (1998)	0.8412744	7
Casablanca (1942)	0.8077428	11232
Rear Window (1954)	0.8058817	7935
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	0.8025903	2922
Third Man, The (1949)	0.7981535	2967

title	b_i	n
Double Indemnity (1944)	0.7972415	2154

We can see that the regularized movie effect makes more sense looking at the titles of the movie names.

Lets look at the least 10 movies with lowest ratings as well.

```
train_set %>%
  dplyr::count(movieId) %>%
  left_join(movie_reg_avgs) %>%
  left_join(movie_titles, by="movieId") %>%
  arrange(b_i) %>%
  select(title, b_i, n) %>%
  slice(1:10) %>%
  knitr::kable()
```

```
## Joining, by = "movieId"
```

title	b_i	n
SuperBabies: Baby Geniuses 2 (2004)	-2.579628	56
From Justin to Kelly (2003)	-2.571686	199
Pokémon Heroes (2003)	-2.430055	137
Disaster Movie (2008)	-2.425683	32
Carnosaur 3: Primal Species (1996)	-2.321798	68
Glitter (2001)	-2.316449	339
Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002)	-2.300088	202
Gigli (2003)	-2.297157	313
Barney's Great Adventure (1998)	-2.291909	208
Hip Hop Witch, Da (2000)	-2.216148	14

Looking at the movie titles it is understood why these movies were given lower ratings.

Get the predicted ratings based on this regularized movie effect model.

```
predicted_ratings <- test_set %>%
  left_join(movie_reg_avgs, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  .$pred

# Calculate RMSE and add to the results table
model_3_rmse <- RMSE(predicted_ratings, test_set$rating)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie Effect Model",
                                     RMSE = model_3_rmse ))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0603313
Movie Effect Model	0.9439087

method	RMSE
Movie + User Effects Model	0.8292477
Regularized Movie Effect Model	0.9438538

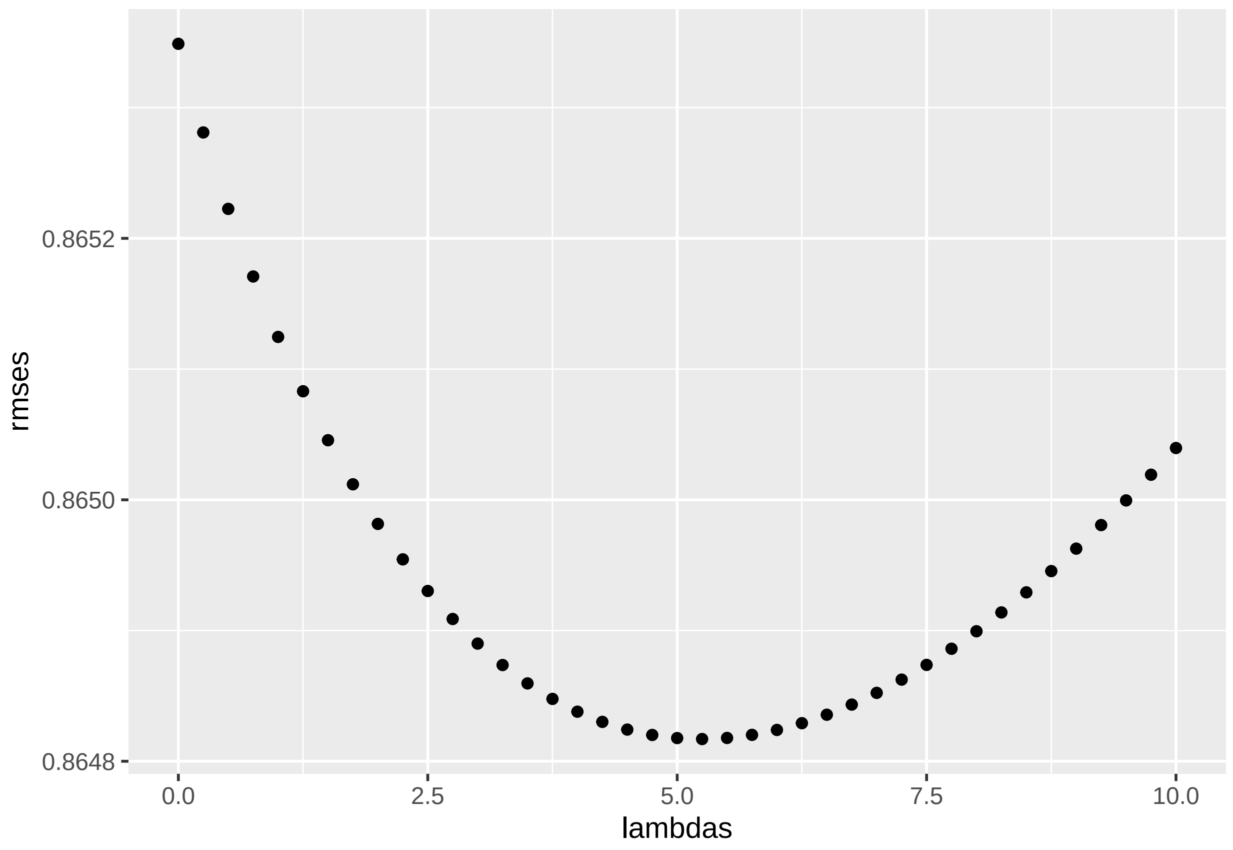
Comparing the RMSE for the regularized movie effect model with the original movie effect model it doesn't appear regularization improved RMSE siginificantly. I checked other values for lambda like 2, 6, 100, 1000 - I get similar RMSE results with those values too.

Lets now build the regularized movie and user effect model. This time we will use different values for lambda to tune it for the best RMSE result.

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

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

```
# Lets plot to see the lambda that gives the least RMSE
qplot(lambdas, rmses)
```



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

```
## [1] 5.25
```

5.25 is the lambda that gives the minimum RMSE as seen above.

Lets add it to our RMSE table

```
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie + User Effect Model",
                                     RMSE = min(rmses)))
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0603313
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8292477
Regularized Movie Effect Model	0.9438538
Regularized Movie + User Effect Model	0.8648170

We can see a considerable improvement with the regularized movie and user effect model. Note that this is already lower than the target RMSE of 0.86490. But we will try the Matrix Factorization method to see if there is an improvement.

Next we will try the final model - Matrix Factorization - we will use the Reco R package to build a recommender system using Matrix Factorization. First create a data frame with user, movie and rating.

```r
train_set <- edx  %>%  select(userId, movieId, rating) %>%
  mutate(userId = as.integer(userId)
         ,movieId = as.integer(movieId)
         ,rating = rating)

# Lets specify the source of data to the recommender system. This is stored
# in memory rather than refering to a file.
train_Memory <- data_memory(user_index = train_set$userId,
                            item_index = train_set$movieId,
                            rating = train_set$rating, index1 = TRUE)

# Create an instance of the recommender system and initialize it
recommender <- Reco()
```

Next, tune the recommender system. The following parameters are used to tune the recommendation system to ensure this does not take a lot of time to run. Most are just the first default value except for the number of iterations. I set the no. of iterations to 10 because I found this is enough to give a considerable improvement in RMSE. Other values for iterations do not appear to result in a noticeable change to the RMSE.

```r
opts <- recommender$tune(
  train_Memory,
  opts = list(
    dim      = c(70), # number of latent factors
    costp_l1 = 0, # L1 regularization cost for user factors
    costq_l1 = 0, # L1 regularization cost for item factors
    nthread  = 4, # number of threads for parallel computing
    niter    = 10 # number of iterations
  )
)
```

Train the recommender system with optimized parameters.

```r
recommender$train(train_Memory, opts = c(opts$min, # optimized parameters
                                         niter = 10, # iterations
                                         nthread = 4)) # number of threads
```

```
## iter      tr_rmse          obj
##    0       1.7070   2.6258e+07
##    1       1.9642   3.4987e+07
##    2       2.1064   4.0625e+07
##    3       2.0025   3.7114e+07
##    4       1.8958   3.3513e+07
##    5       1.8537   3.2161e+07
##    6       1.8550   3.2241e+07
##    7       1.8704   3.2789e+07
##    8       1.8872   3.3370e+07
##    9       1.9098   3.4136e+07
```

Prepare the validation set to calculate RMSE.

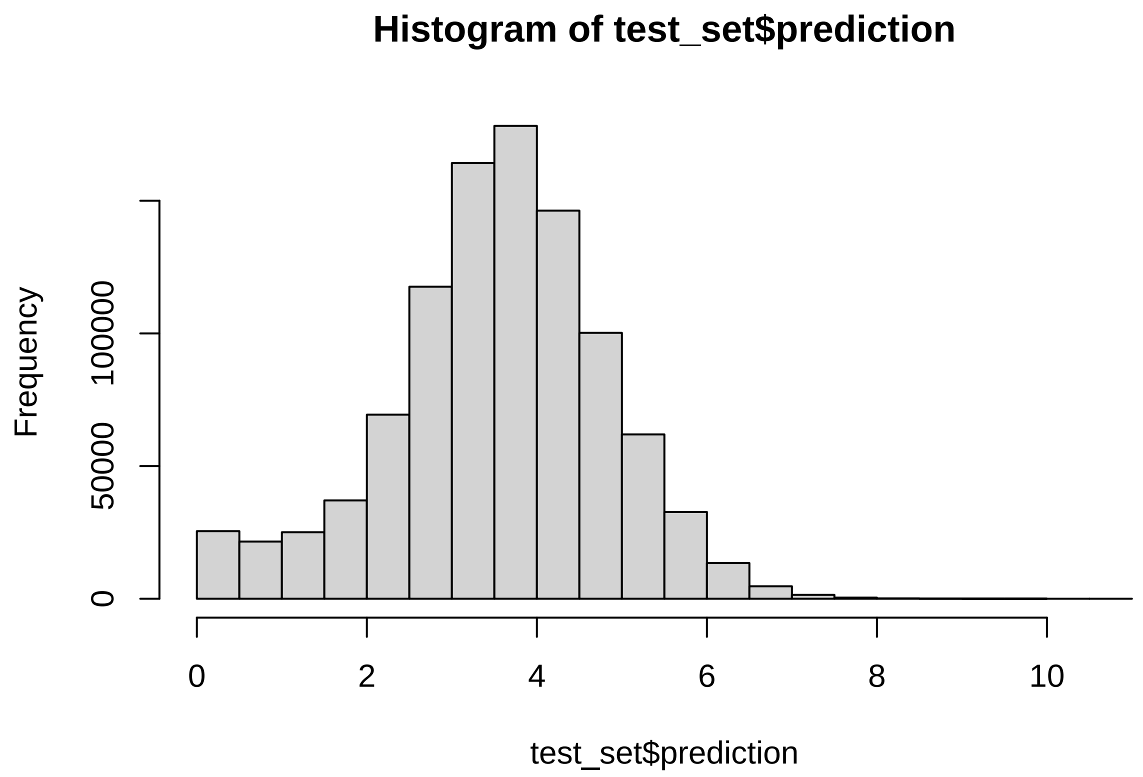
```
test_set <- validation %>% select(userId, movieId, rating) %>%
  mutate(userId = as.integer(userId)
         ,movieId = as.integer(movieId)
         ,rating = rating)
test_Memory <- data_memory(user_index = test_set$userId,
                           item_index = test_set$movieId,
                           rating = test_set$rating, index1 = TRUE)
```

Get prediction for validation data set.

```
test_set$prediction <- recommender$predict(test_Memory, out_memory())
```

Lets make a histogram of the predictions to see the distribution of the predicted ratings.

```
hist(test_set$prediction)
```



We can see that there are some values below 0.5:

```
sum(test_set$prediction < 0.5)
```

```
## [1] 25474
```

There are also some values above 5:

```
sum(test_set$prediction > 5)
```

```
## [1] 114933
```

We can see there are a few values that are greater than 5 and less than 0.5 rating. We will set these to 5 and 0.5 respectively.

```
test_set <- test_set %>% mutate(prediction_opts = case_when(prediction < 0.5 ~ 0.5, prediction > 5 ~ 5,
```

Calculate RMSE and record result

```
RMSE <- RMSE(test_set$rating, test_set$prediction_opts)
# When input is the original, real-valued rating use: RMSE <- rmse(test$rating, ceiling(test$prediction
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Matrix Factorization Model",
                                     RMSE = RMSE))
```

Results

Here are the results for the six models we used for our prediction along with the RMSE results:

```
rmse_results %>% knitr::kable()
```

method	RMSE
Just the average	1.0603313
Movie Effect Model	0.9439087
Movie + User Effects Model	0.8292477
Regularized Movie Effect Model	0.9438538
Regularized Movie + User Effect Model	0.8648170
Matrix Factorization Model	1.3328948

As can be seen the most effective model is Matrix Factorization. The movie and user effect appears to have been captured efficiently by the Reco recommender system using Matrix Factorization. The next best model is the regularized movie and user effects model which seems to have a considerable improvement over the original movie and user effects model.

Conclusion

This project tested five (six including the average model) machine learning models in order to recommend movie ratings based on the movie and user characteristics. One problem with this prediction is that we could see that rating 4.0 was considerably more popular than the other ratings. As a result the models may have been biased more towards the higher ratings and thus could be less accurate with other datasets. We can check the specificity and sensitivity of the model by forcing it to predict only multiples of 0.5 from 0 to 5.