MovieLens Project Submission

Michael Krog

June 2019

Introduction

This document serves as an analysis of using different prediction models on the MovieLens database in an attempt to ascertain an RMSE of less than 0.87750 for predicting the rating any given user is likely to give to a movie.

Method/Analysis

A step by step approach has been used in comparing different methods of prediction against each other whilst adding new forms of bias. For example I have begun the analysis by creating a prediction based simply on the average rating per movie, and then have attempted to refine this result moving forward.

The methods included in this analysis are as follows: 1) "Just the average" A prediction based on the average of all movie ratings 2) "Movie effect model" A prediction including a movie bias. I.e. taking into account the changes in rating per movie. 3) "Movie + user effects model" A prediction adding user bias to movie bias. I.e. taking into account that different users give different ratings on average. 4) "Regularised movie + user effect model" A prediction using regularization to account for movies with very few ratings potentially skewing results.

The database used is a subset of the MovieLens database, and has been split into a training 'edx' set and a testing 'validation' set which has been used for the final test of the above listed approaches to determine the most effective in reducing the RMSE.

Note that all RMSE calculations have been performed on the validation set.

Before testing these approaches the data was downloaded and split into these two sets. This was followed by data cleaning and data exploration. Visualisation of the data has been used throughout the report where relevant to visualise a finding or help validate a point.

The results and conclusion of this analysis as well as the recommendation of the best approach can be found at the end of the document.

Downloading and cleaning the data

```
# Creating the test and validation sets

####################################
# Create edx set and validation set
####################################

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")

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
```

```
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- read.table(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")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")

# Validation set will be 10% of MovieLens data

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)
```

Exploring the data set

```
#Count of the number of rows & columns

nrow(edx)
```

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

```
ncol(edx)
```

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

```
#Number of 0 & 3 ratings

length(which(edx$rating == 0))
```

```
## [1] 0
```

```
length(which(edx$rating == 3))
```

```
## [1] 2121240
```

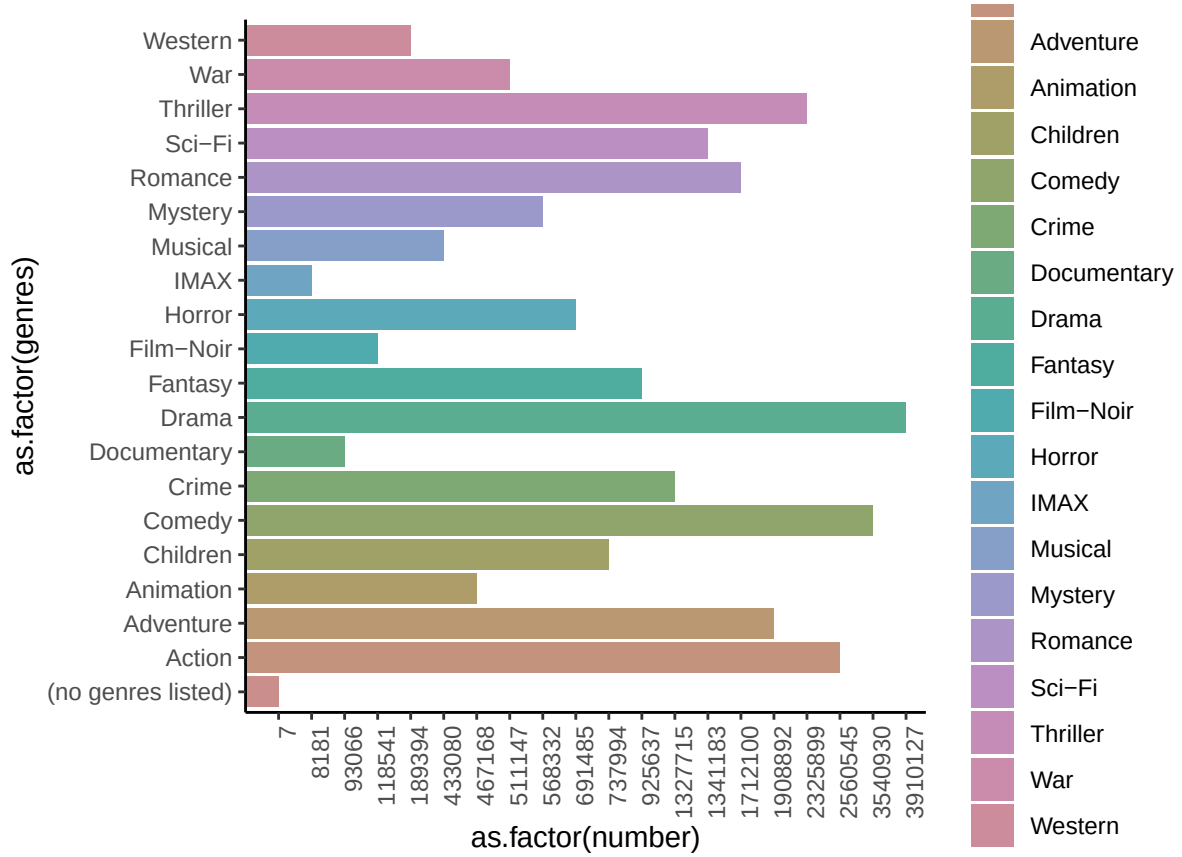
```
#Number of different movies

diff_movies <- unique(edx$movieId)
length(diff_movies)
```

```
## [1] 10677
```

```
#Number of different users

n_distinct(edx$userId)
```

```
## [1] 69878
```

```
#Number of movie ratings in each genre

genres <- edx %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarise(number = n()) %>%
  arrange(desc(number))

ggplot(genres, aes(x=as.factor(genres), y=as.factor(number), fill = as.factor(genres))) +
  geom_bar(stat = "identity") +
  theme_classic() +
  scale_fill_hue(c =40) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  coord_flip()
```



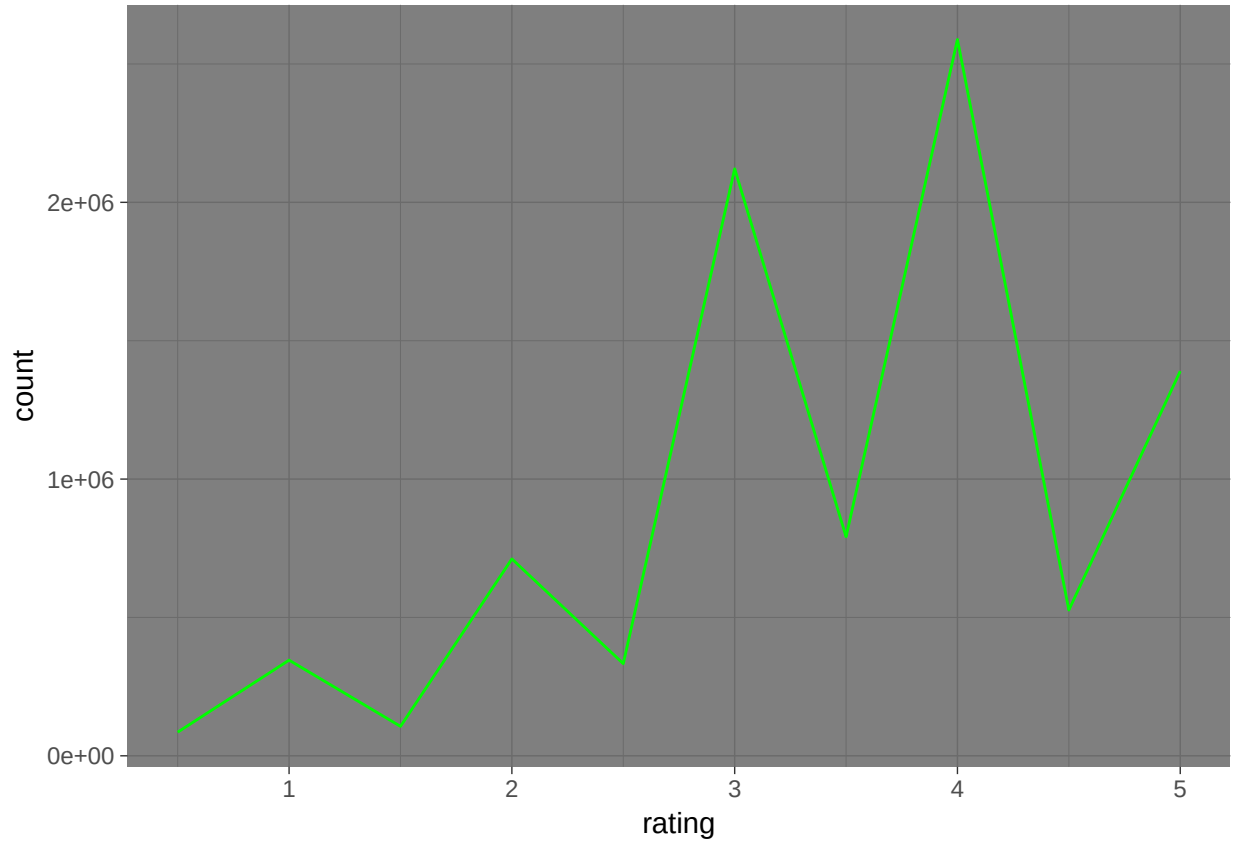

```
#Which movie has most ratings:

edx %>% group_by(title) %>% summarise(number = n()) %>%
  arrange(desc(number))
```

```
## # A tibble: 10,676 x 2
##    title                                                     number
##    <chr>                                                      <int>
##  1 Pulp Fiction (1994)                                        31362
##  2 Forrest Gump (1994)                                        31079
##  3 Silence of the Lambs, The (1991)                           30382
##  4 Jurassic Park (1993)                                       29360
##  5 Shawshank Redemption, The (1994)                           28015
##  6 Braveheart (1995)                                          26212
##  7 Fugitive, The (1993)                                       25998
##  8 Terminator 2: Judgment Day (1991)                          25984
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)  25672
## 10 Apollo 13 (1995)                                           24284
## # ... with 10,666 more rows
```

```
#What are the five most given ratings in order from most to least?

edx %>% group_by(rating) %>%
  summarise(number = n()) %>%
  arrange(desc(number))
```

```
## # A tibble: 10 x 2
##    rating  number
##     <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
```

```
#.5 star ratings more or less common than whole?
# (Taken directly from the course work)

edx %>% group_by(rating) %>%
  summarise(count = n()) %>%
  ggplot(aes(x = rating, y = count)) +
  theme_dark() +
  geom_line(colour = "Green")
```



Creating the calculation for RMSE

Having explored the data I now want to set up my equation that will be used for calculating RMSE which is below:

Creating the RMSE function

```
#Create RMSE function

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

Step 1: Building the simplest possible recommendation system

Using a model based approach assuming the same ratings for all movies and users with all differences explained by random variation.

Running the RMSE function against the average rating for all movies

```
#Avg rating of all movies across all users:

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

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

```
#How well does this prediction do on the test set?

naive_rmse <- RMSE(validation$rating, mu_hat)
naive_rmse
```

```
## [1] 1.061202
```

This approach produced an RMSE of 1.061 which is far from our goal of 0.8775 so we will continue to try other approaches.

To display the differences in approaches as I progress a table will be added to indicate results through progression:

Results table - 'Just the average'

```
#To show continued improvements in the RMSE a table will be created as below:

rmse_results <- data_frame(method = "Just the average", RMSE = naive_rmse)

rmse_results
```

```
## # A tibble: 1 x 2
##   method            RMSE
##   <chr>            <dbl>
## 1 Just the average  1.06
```

So, how can we improve our RMSE further?

To do this we need to look at what could have an effect on the RMSE which we aren't taking into account when calculating the average rating.

An obvious example is that every moving will have different ratings and different rating averages.

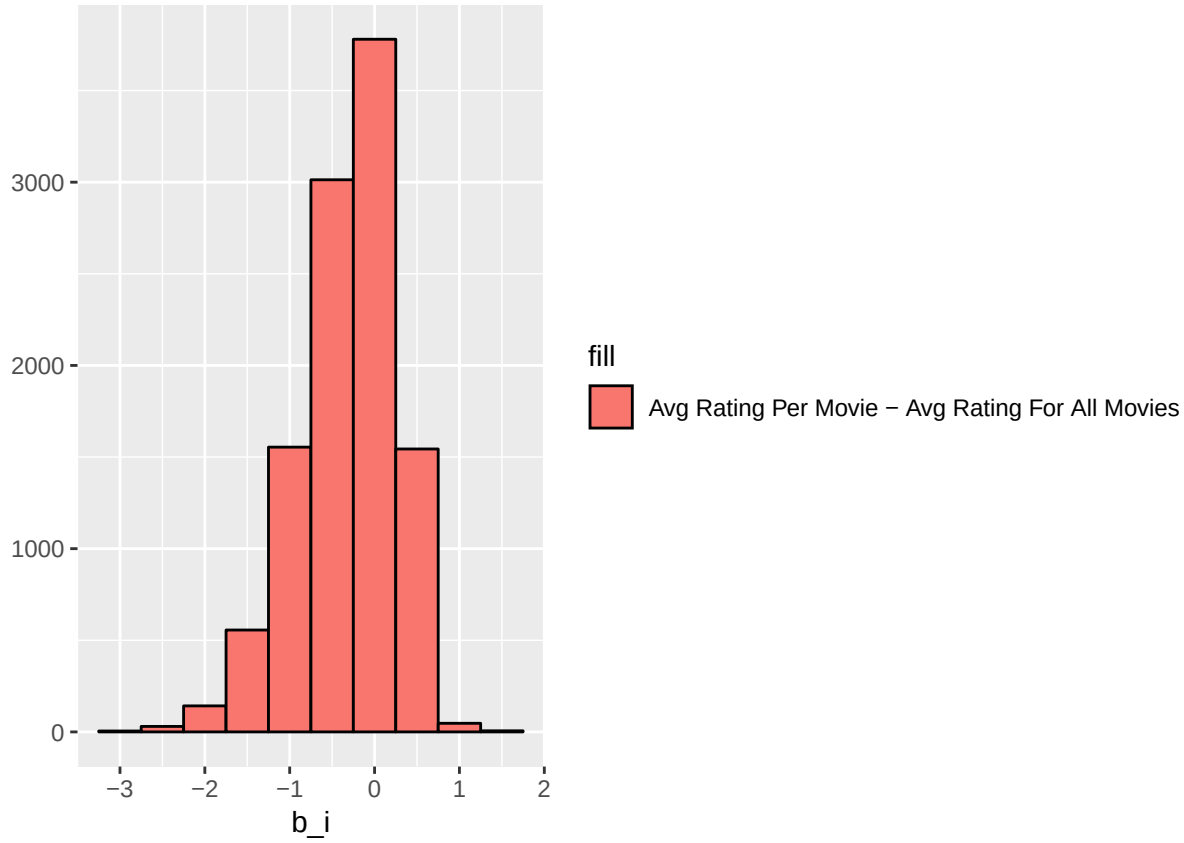
Therefore for the next step we are going to introduce a consideration for movie bias.

Movie Effect Model

In the Movie Effect Model, I've added in a movie bias using the mean rating of different movies and then subtracted the average rating across all movies as can be seen below:

Showing the movie bias

```
mu <- mean(edx$rating) #Create avg rating across all movies.
movie_avgs <- edx %>% #Create avg rating per movie from the edx data set.
  group_by(movieId) %>% #Group by movie ID.
  summarise(b_i = mean(rating - mu)) #b_i is the average rating for any given movie minus the overall a

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




```
#movies with a b_1 1.5 = 5 star avg movie ratings!
```

To see if adding in this bias is able to improve the prediction I have run the below model.

Running the 'Movie effect model' and showing the results

```
predicted_ratings <- mu + validation %>% #Create the column of predicted ratings.
     left_join(movie_avgs, by='movieId') %>%
     .$b_i

model_1_rmse <- RMSE(predicted_ratings, validation$rating) #Test the predicted ratings against the vali
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie effect model",
                                     RMSE = model_1_rmse )) #Table the RMSE results.
rmse_results %>% knitr::kable() #Show in table.
```

method	RMSE
Just the average	1.0612018
Movie effect model	0.9439087

From the above it can be seen that the RMSE has improved from 1,06 to 0.94. But this is still quite far

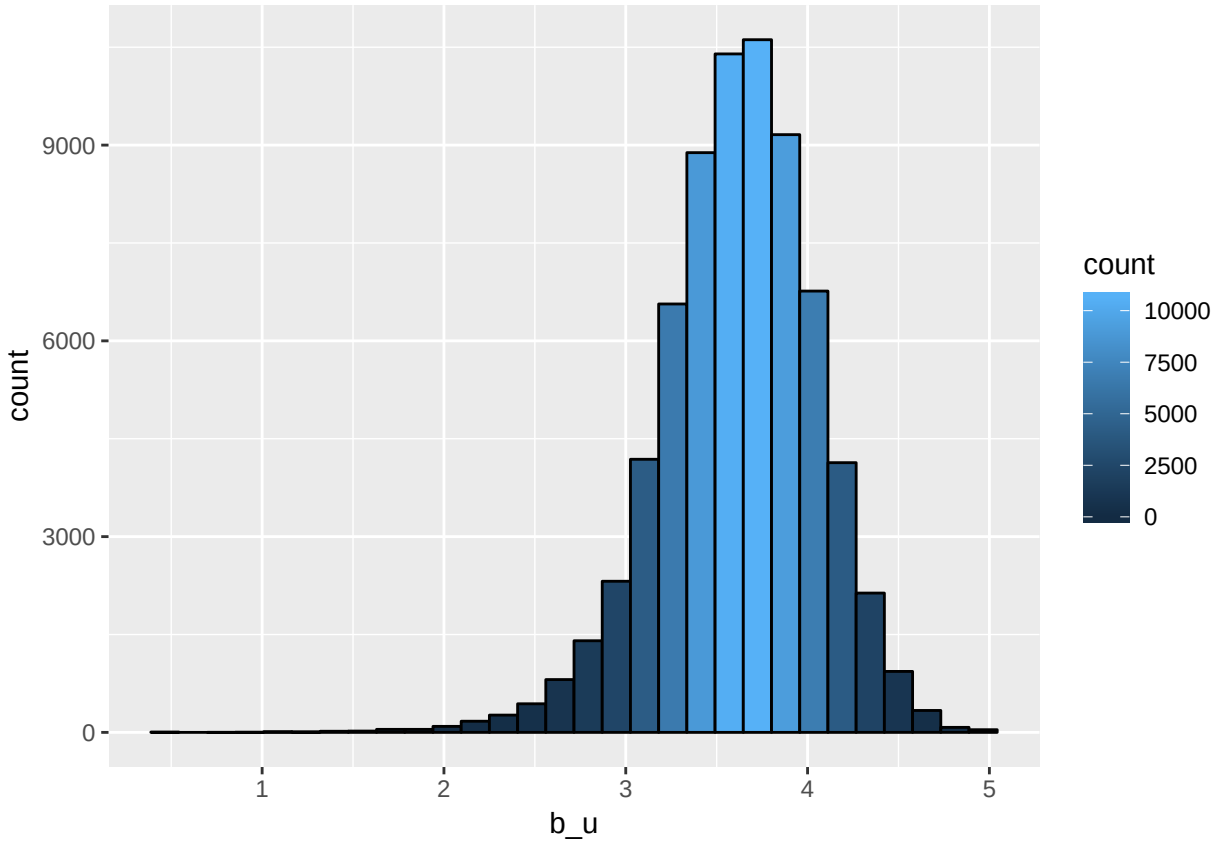
from the goal of 0.8775.

The next step is to add in a further factor into the prediction model - users. Do some users rate movies better on average than other users?

Movie + user effect model

Showing avg ratings per user

```
#Avg rating per user for users who have rated over 100 movies

edx %>%
     group_by(userId) %>%  #Group by user ID
     summarize(b_u = mean(rating)) %>% #Create avg ratings per user
     filter(n()>=100) %>% #Filter the top 100 results
     ggplot(aes(b_u)) + #Plot avg ratings per user
     geom_histogram(bins = 30, color = "black", aes(fill = ..count..))
```



From the above histogram we can see that some users on average give higher ratings, where some users give lower ratings and the average user rating sitting somewhere just below 4.

This bias is now added on top of the movie bias to identify if this will help improve the likelihood of our prediction being correct.

Including user bias in the prediction and showing the results

```
user_avgs <- edx %>% #Create user_avgs from the edx data set.
     left_join(movie_avgs, by='movieId') %>% #Join with movie_avgs by user ID.
     group_by(userId) %>% #Group by user ID.
     summarize(b_u = mean(rating - mu - b_i)) #Avg unique user rating = avg of the avg rating minus the

predicted_ratings2 <- validation %>% #Create the predicted ratings from the validation set.
     left_join(movie_avgs, by='movieId') %>% #Join movie_avgs by movie ID.
     left_join(user_avgs, by='userId') %>% #Join user_avgs by user ID.
     mutate(pred = mu + b_i + b_u) %>% #Create predictions combining avg rating + movie bias + user bia
     .$pred

model_2_rmse <- RMSE(predicted_ratings2, validation$rating) #Run the RMSE prediction model.
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie + user effects model",  #Create the rmse_results dat
                                     RMSE = model_2_rmse ))
rmse_results %>% knitr::kable() #Plot results into table.
```

method	RMSE
Just the average	1.0612018
Movie effect model	0.9439087
Movie + user effects model	0.8653488

As can be seen above the RMSE has now decreased to 0.86. This is surprising as it achieves our goal of 0.8775 quite easily.

To be honest this is not the result from this approach I would have expected. However given its success I have continued into further methods to ascertain if another approach can help lower the RMSE further still.

To do this I'm going to use regularisation to attempt reducing the RMSE even further.

Regularised movie + user effect model

When looking at the original 'Movie effect model', there is the question of whether this can be improved further.

To do this I've explored the mistakes that were made in this initial assumption

To start I've looked at the 10 largest mistakes that were made. These can be seen in the below:

The biggest mistakes made in the 'Movie effect model'

```
#Looking at 10 largets errors of movie effects only:

edx %>% #Using the edx data set.
     left_join(movie_avgs, by='movieId') %>% #Joining movie_avgs by movie ID.
     mutate(residual = rating - (mu + b_i)) %>% #Creating 'redidual column of rating minus the sum of a
     arrange(desc(abs(residual))) %>% #Arranging residual in descending order.
     select(title,  residual) %>% slice(1:10) %>% knitr::kable() #Show first ten results of 'title' and
```

title	residual
From Justin to Kelly (2003)	4.097990
From Justin to Kelly (2003)	4.097990
Pokémon Heroes (2003)	3.970803
Pokémon Heroes (2003)	3.970803
Shawshank Redemption, The (1994)	-3.955131
Shawshank Redemption, The (1994)	-3.955131
Shawshank Redemption, The (1994)	-3.955131
Shawshank Redemption, The (1994)	-3.955131
Shawshank Redemption, The (1994)	-3.955131
Shawshank Redemption, The (1994)	-3.955131

Next I've looked at what the 10 best & 10 worst movies were according to the estimates:

Top 10 best movies based on original 'Movie effect method'

```
#10 best movies based on estimates:

movie_titles <- edx %>% #Create 'movie_titles' using edx dataset.
     select(movieId, title) %>% #Select movie ID and title columns.
     distinct() #Keep unique rows
movie_avgs %>% left_join(movie_titles, by="movieId") %>% #Join 'movie_titles' to 'movie_avgs by movie I
     arrange(desc(b_i)) %>% #Arrange in descending order according to movie bias
     select(title, b_i) %>% #Select the title and movie bias columns
     slice(1:10) %>% #Choose the first ten entries
     knitr::kable() #Place into table
```

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

Top 10 worst movies based on original 'Movie effect method'

```
#10 worst movies based on estimates:

movie_avgs %>% left_join(movie_titles, by="movieId") %>% #Join 'movie_titles' to 'movie_avgs by movie I
     arrange(b_i) %>% #Arrange by movie bias
     select(title, b_i) %>% #Select the title and movie bias columns
     slice(1:10) %>% #Choose the first ten entries
```

```
    knitr::kable() #Place into table
```

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

These are all obscure movies so why is this the case? It may be because they have only receive one or two ratings skewing the data and not providing a good avg. The below exploration shows that this is the case:

10 worst movies including the count of ratings for each movie:

```
#Creating the same table but now including the number of ratings received.

#10 worst movies:

edx %>% dplyr::count(movieId) %>% #Count ratings per movie ID.
    left_join(movie_avgs) %>% #Join to 'movie_avgs'.
    left_join(movie_titles, by="movieId") %>% #Join 'movie_titles' by movie ID.
    arrange(desc(b_i)) %>% #Arrange in descending order by movie bias.
    select(title, b_i, n) %>% #Select the title, movie bias, and number of ratings columns.
    slice(1:10) %>% #Show first ten entries.
    knitr::kable() #Show in table.
```

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

10 best movies with the count of ratings for each movie:

```
#10 best movies:
edx %>% dplyr::count(movieId) %>% #Count ratings by movie ID.
     left_join(movie_avgs) %>% #Join to 'movie_avgs'.
     left_join(movie_titles, by="movieId") %>% #Join 'movie_titles' by movie ID.
     arrange(b_i) %>% #Arrange by movie bias.
     select(title, b_i, n) %>% #Select the title, movie bias, and number of ratings columns.
     slice(1:10) %>% #Show first ten entries.
     knitr::kable() #Show in table.
```

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

You can see that the supposed best and worst movies were only rated by very few users.

With fewer user ratings we have much higher uncertainty so these are essentially 'noisy' estimates that we shouldn't trust in the prediction.

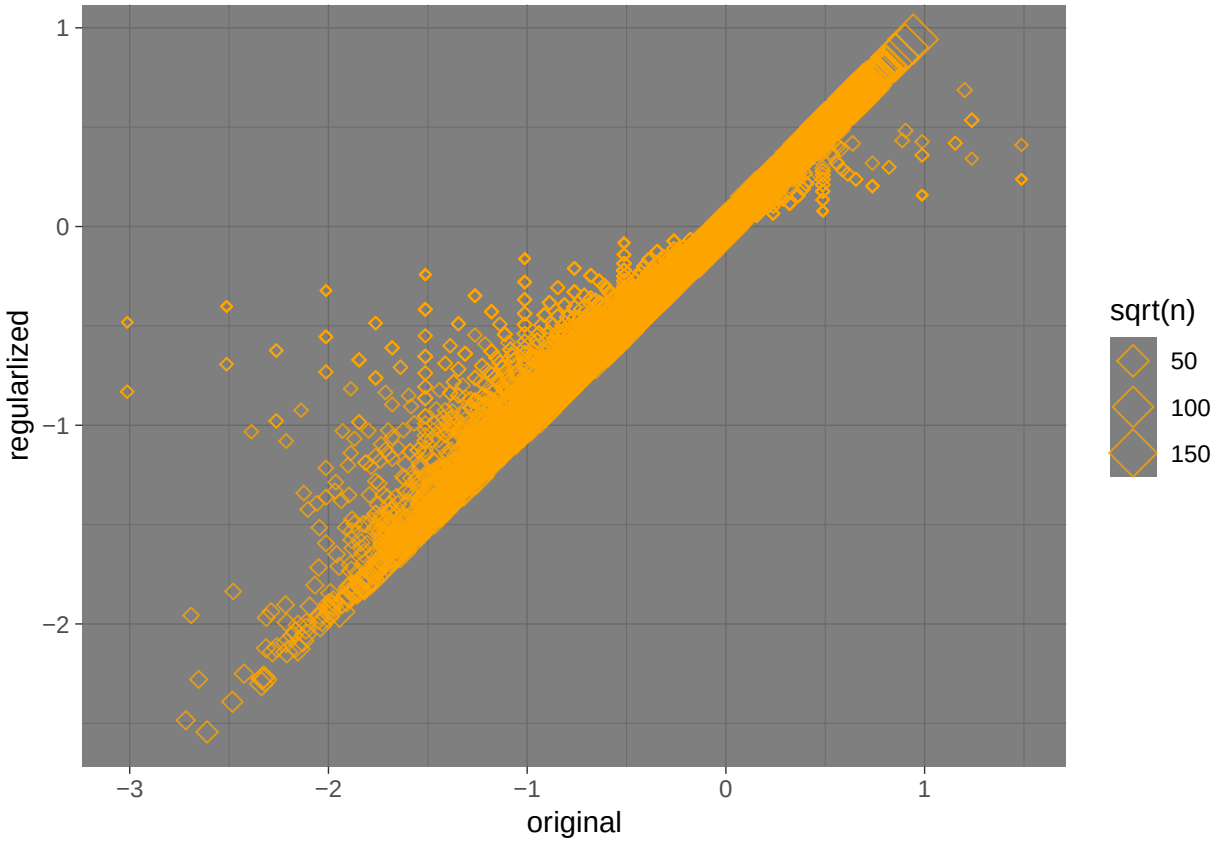
To penalise large estimates such as those shown above I've used regularisation.

I've used lambda where the larger lambda the more we shrink closer to zero.

I've used lambda = 5.25 and demonstrate why further in the document.

Introducing regularisation

```
#Regularization using lambda = 5.25

lambda <- 5.25 #Create 'lambda'.
mu <- mean(edx$rating) #Create overall avg rating from 'edx'.
movie_reg_avgs <- edx %>%  #Create 'movie_reg_avgs' (movie regularisation averages).
     group_by(movieId) %>% #Group by movie ID.
     summarize(b_i = sum(rating - mu)/(n()+lambda), n_i = n()) #Summarise movie bias (including lambda


#Plot to visualise the differences between original and regularized estimates:

data_frame(original = movie_avgs$b_i, #Create a data frame using the original movie bias and the regula
           regularlized = movie_reg_avgs$b_i,
           n = movie_reg_avgs$n_i) %>%
     ggplot(aes(original, regularlized, size=sqrt(n))) + #Plot to show biases that will be penalised (l
  geom_point(shape=5, alpha=0.75, colour = "Orange") +
  theme_dark()
```



So now let's look at our top 10 best predictions when using regularisation:

Top 10 movies including regularisation effect:

```
edx %>%
     dplyr::count(movieId) %>% #Count ratings by movie ID.
     left_join(movie_reg_avgs) %>% #Join 'movie_reg_avgs'.
     left_join(movie_titles, by="movieId") %>% #Join 'movie_title' by movie ID.
     arrange(desc(b_i)) %>% #Arrange in descending order by movie bias.
     select(title, b_i, n) %>% #Select the title, movie bias and count columns.
     slice(1:10) %>% #Select the first 10 entries.
     knitr::kable() #Show in a table.
```

title	b_i	n
Shawshank Redemption, The (1994)	0.9424894	28015
Godfather, The (1972)	0.9026338	17747
Usual Suspects, The (1995)	0.8531815	21648
Schindler's List (1993)	0.8508355	23193
Casablanca (1942)	0.8075811	11232
Rear Window (1954)	0.8056533	7935
Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	0.8019734	2922
Third Man, The (1949)	0.7975492	2967
Double Indemnity (1944)	0.7964108	2154
Paths of Glory (1957)	0.7936033	1571

The above now makes much more sense in terms of estimating the best movies.

Below is a look at the worst movies based on the estimates using regularisation:

Worst 10 movies including regularisation effect:

```
edx %>%
     dplyr::count(movieId) %>% #Count ratings by movie ID.
     left_join(movie_reg_avgs) %>% #Join 'movie_reg_avgs'.
     left_join(movie_titles, by="movieId") %>% #Join 'movie_title' by movie ID.
     arrange(b_i) %>% #Arrange by movie bias.
     select(title, b_i, n) %>% #Select the title, movie bias and count columns.
     slice(1:10) %>% #Select the first 10 entries.
     knitr::kable() #Show in a table.
```

title	b_i	n
From Justin to Kelly (2003)	-2.543356	199
SuperBabies: Baby Geniuses 2 (2004)	-2.484866	56
Pokémon Heroes (2003)	-2.391618	137
Glitter (2001)	-2.301309	339
Gigli (2003)	-2.280916	313
Disaster Movie (2008)	-2.279165	32
Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002)	-2.275117	202
Barney's Great Adventure (1998)	-2.267727	208
Carnosaur 3: Primal Species (1996)	-2.250480	68
Son of the Mask (2005)	-2.141303	165

Again the above makes much more sense.

Using regularisation has improved the RMSE slightly from the original movie effect model, and this can be seen in the table below:

Prediction results of 'Regularisation + movie effect model'

```
predicted_ratings3 <- validation %>% #Create 'predicted_ratings3 from the validation set.
     left_join(movie_reg_avgs, by='movieId') %>% #Join 'movie_reg_avgs by movie ID.
     mutate(pred = mu + b_i) %>% #Create 'pred' from the overall avg and movie bias.
     .$pred

model_3_rmse <- RMSE(predicted_ratings3, validation$rating) #Run the prediction model against validatio
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Regularized Movie Effect Model", #Create a data frame of t
                                     RMSE = model_3_rmse ))
rmse_results %>% knitr::kable() #Show in a table.
```

method	RMSE
Just the average	1.0612018
Movie effect model	0.9439087

method	RMSE
Movie + user effects model	0.8653488
Regularized Movie Effect Model	0.9438805

You can see by the table above that this has given an very slight improvement on the original 'Movie effect' model.

Now by including the user effect model + the regularized movie effect model we will hopefully have a better RMSE than movie + user effects model, as can be seen below:

This is achieved using lambda as a tuning parameter, where the best number for lambda is selected from between 0 and 10 at intervals of 0.25. This tuning is where the figure of 5.25 is found as the optimal representation of lambda as was used previously in the document to penalise large estimates based on few ratings.

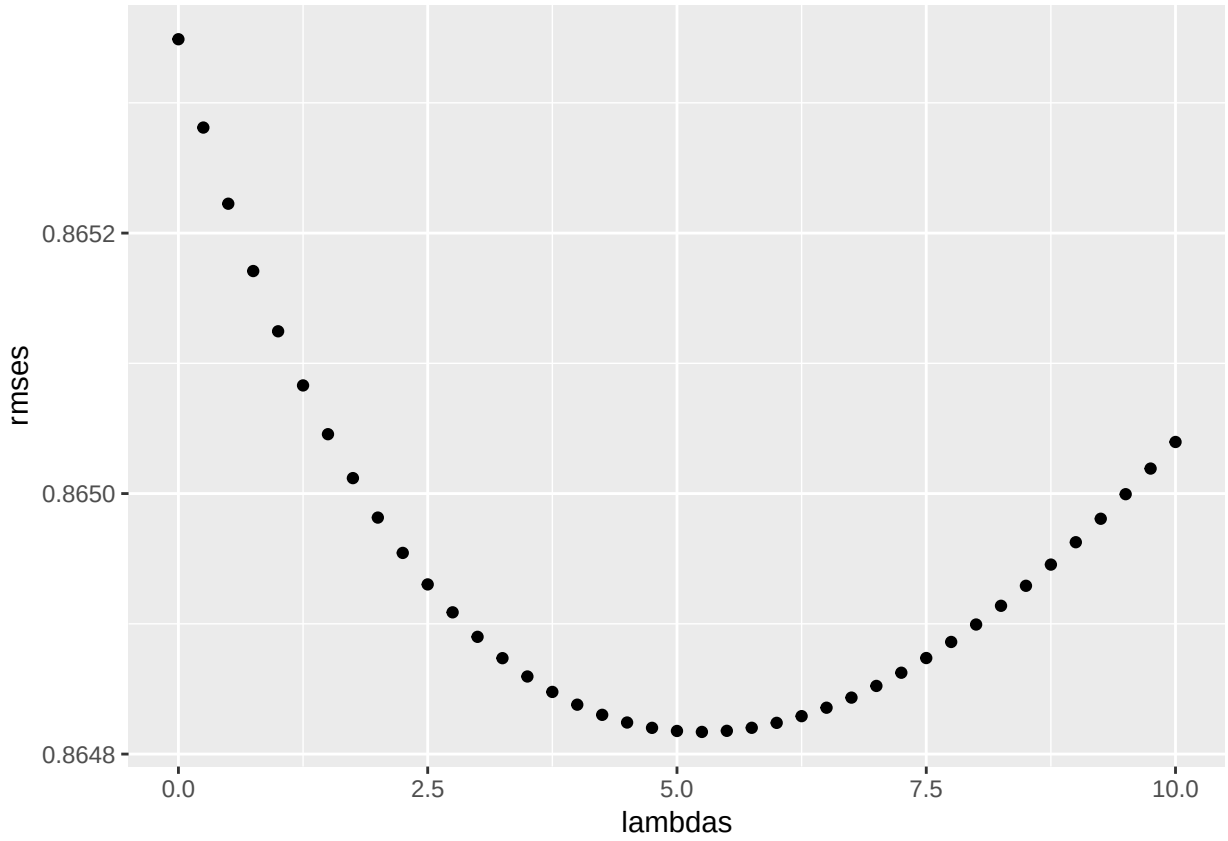
This tuning is included below in the predictions of 'Regularized movie + user effects' model.

Introducing tuning through 'lambda'

```
lambdas <- seq(0, 10, 0.25) #Setting tuning parameters for finding best lambda.
rmses <- sapply(lambdas, function(l){ #Apply lambdas to the function below.
     mu <- mean(edx$rating) #Creating the overall avg rating
     b_i <- edx %>% #Creating the movie bias
          group_by(movieId) %>%
          summarize(b_i = sum(rating - mu)/(n()+l)) #Create movie bias.
     b_u <- edx %>% # Creating the user bias as previously shown
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu)/(n()+l)) #Create user bias.
     predicted_ratings <- #Creating the prediction from the validation set.
          validation %>%
          left_join(b_i, by = "movieId") %>%
          left_join(b_u, by = "userId") %>%
          mutate(pred = mu + b_i + b_u) %>% #Create pred from overall avg, movie bias and user bias (la
          .$pred
     return(RMSE(predicted_ratings, validation$rating)) #Run the RMSE function.
})
```

Plot to show best lambda to produce lowest RMSE

```
qplot(lambdas, rmses) #Plot lambdas vs RMSE to show best lambda for RMSE
```




```
# Lambda with the lowest RMSE:

lambda <- lambdas[which.min(rmses)] #Select the lambda that gives the lowest RMSE.
lambda
```

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

Run predictions and table results

```
rmse_results <- bind_rows(rmse_results, #Create data frame to show RMSE.
                          data_frame(method="Regularized movie + user effect model",
                                     RMSE = min(rmses))) #Select the min RMSE from the data frame.
rmse_results %>% knitr::kable() #Show results in table.
```

method	RMSE
Just the average	1.0612018
Movie effect model	0.9439087
Movie + user effects model	0.8653488
Regularized Movie Effect Model	0.9438805
Regularized movie + user effect model	0.8648170

Results

The overall results are as per the table above where we were able to achieve below the goal RMSE of 0.8775 using two different methods. The results showed that running the 'Movie + user effects model' allowed me to reach the goal RMSE, however it was not the best RMSE achieved.

By factoring in regularisation both the 'Movie effect model', and the 'Movie + user effect model' were slightly improved from running these models without regularisation. This means that the 'Regularized movie + user effect model' gave the lowest RMSE of 0.8648.

Conclusion

In conclusion based on having given the lowest RMSE of 0.8648 and surpasssing the target of 0.8775, the 'Regularized movie + user effect model' is our most effective method of predicting user ratings.