MLProject

Mandy Simpson

10/10/2019

Introduction

Netflix is a streaming service, enabling millions of users worldwide to access a large catalogue of movies on demand. Users are able to rate movies that they have watched using a star system of one star (low) to five stars (high), and are presented with personalised recommendations for new movies to watch.

This project, building a "recommendation system" is part of the HarvardX Professional Certificate in Data Science Capstone course. It looks at how those recommendations are made, and in particular whether machine learning techniques can be used to improve predictions of the rating that a individual user will give to a particular movie.

For this project a dataset of 10 million ratings (the MovieLens 10M Dataset) is used to enable the necessary calcuations to be run on an average home computer. This is still a large dataset and some delays may occur when running the code.

The dataset is split into training and validation subsets. The training subset is used to create a range of machine learning algorithms which are then tested on the validation subset.

Success is measured by considering the difference between the ratings predicted by each algorithm, and the actual ratings contained in validation subset. Root Mean Squared Error (RMSE) is used as a single measure of this difference. RMSE takes the square of the error for each prediction, creates an average of these, and then takes the square root of that average. A lower RMSE indicates a more accurate model.

Methods and Analysis

Installing required packages and downloading the data

The code below for installing the required R packages and downloading the MovieLens 10M dataset was provided as part of the course materials.

```
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")

# 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)
```

Data cleaning

The download contains three files, movies.dat, ratings.dat and tags.dat. Only the first two have been used for this project.

The following code was also provided as part of the course materials. This extracts the movies and ratings data, provides appropriate column headings for each, assigns numeric or character classes as needed, and joins the two tables into a single dataframe called `movielens`.

```
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")
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")
```

Creating training and validation sets

The movielens dataset is now split into training and validation subsets, with the validation set being approximately 10% of the total. The training set is named `edx` and the validation set is named `validation`.

Following the initial split any rows which are included in the `validation` subset, but with users or movies which are not also found in the `edx` subset are removed. These are then placed back into `edx`.

Finally all objects created by this download and cleaning process, but which are not required going forward, are removed.

This code was provided as part of the course materials.

```
# Validation set will be 10% of MovieLens data

set.seed(1, sample.kind="Rounding")
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)
```

Inspecting and exploring the datasets

Both the `edx` and `validation` datasets include the following columns:

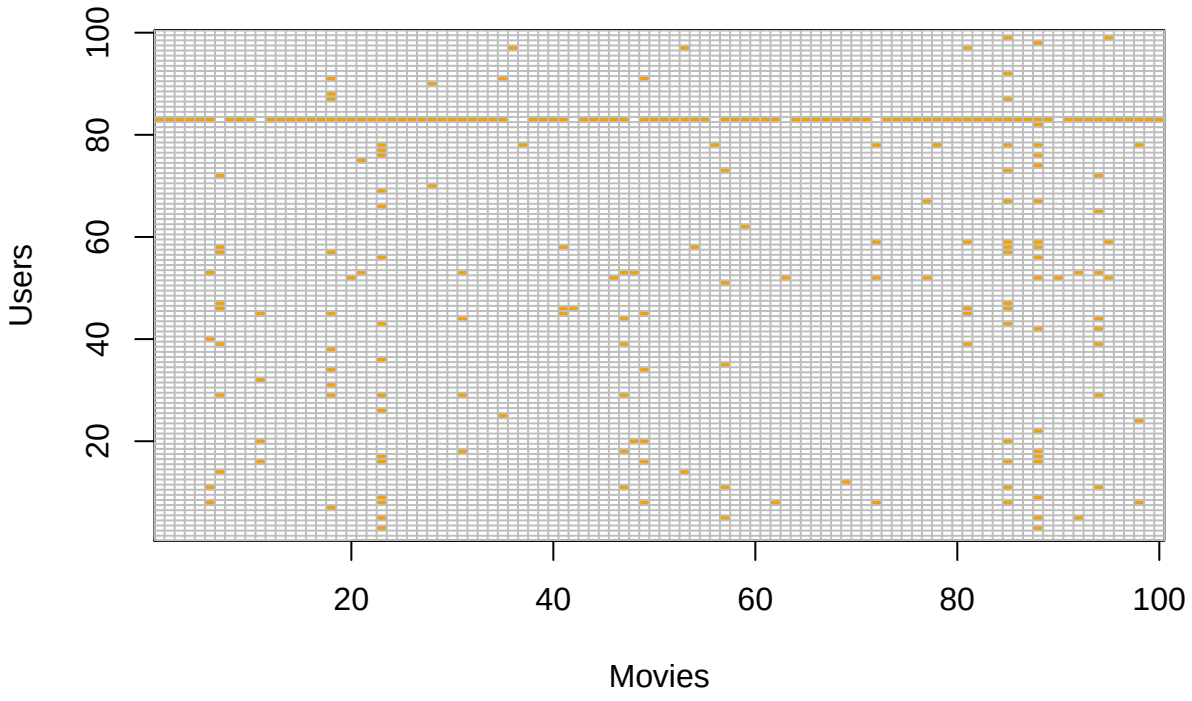
Name	Type	Contains
userId	integer	Unique identification number for each user
movieId	numeric	Unique identification number for each movie
rating	numeric	Star rating from 1(low) to 5(high)
timestamp	integer	Date / time in UNIX timestamp format
title	character	Movie title including release date in brackets
genres	character	Movie genre(s) - multiple genres allowed for each movie

The `edx` dataset has 9,000,055 rows, each representing a single rating by one user of one movie.

Within the `edx` dataset there are 69,878 individual users and 10,677 individual movies represented. This gives a potential total of 746,087,406 possible ratings. However, as seen above, only 9,000,055 ratings are included, just 1.21% of the possible total.

If we represented this data as a table with users as rows and movies as columns, some cells would have ratings but the vast majority would be empty. The task of a recommendation system is, in essence, completing the blank cells.

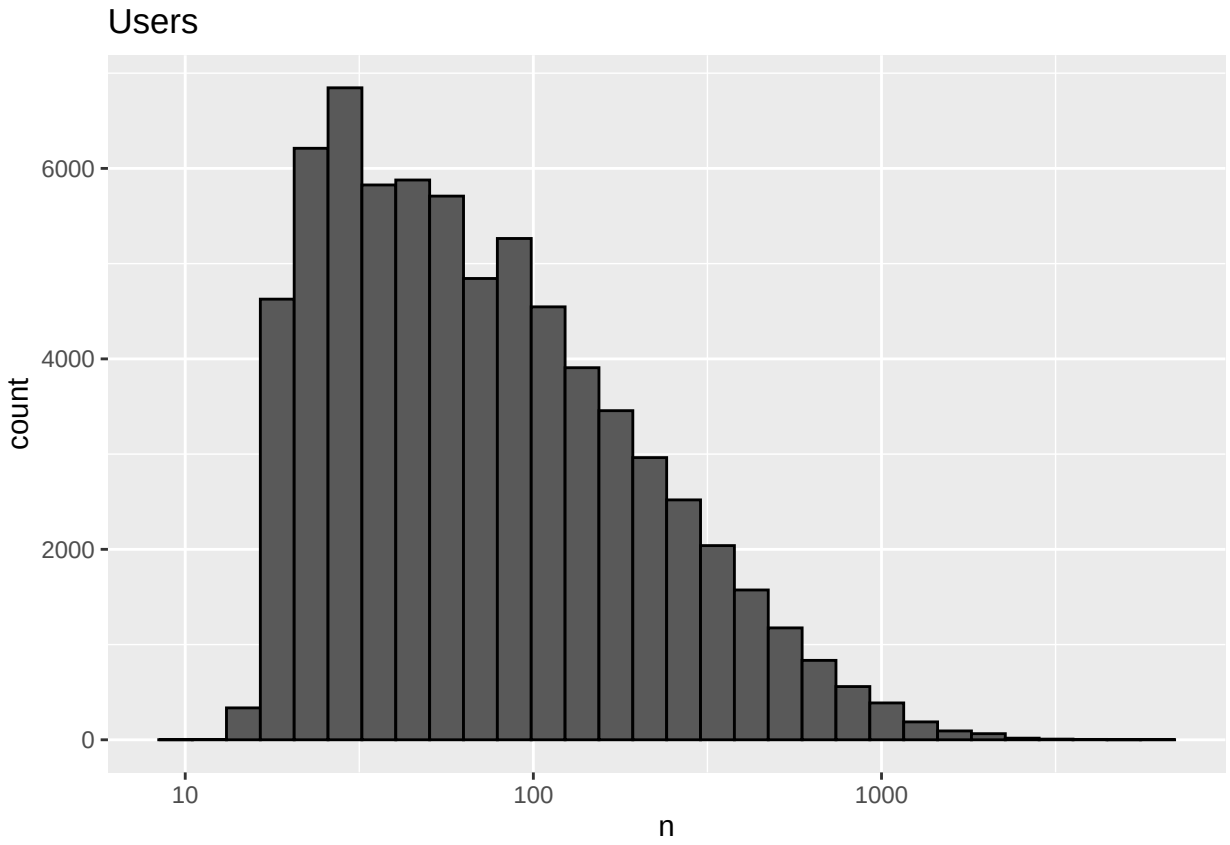
Below is this table for a sample (at random) of 100 users and 100 movies from the `edx` training set:



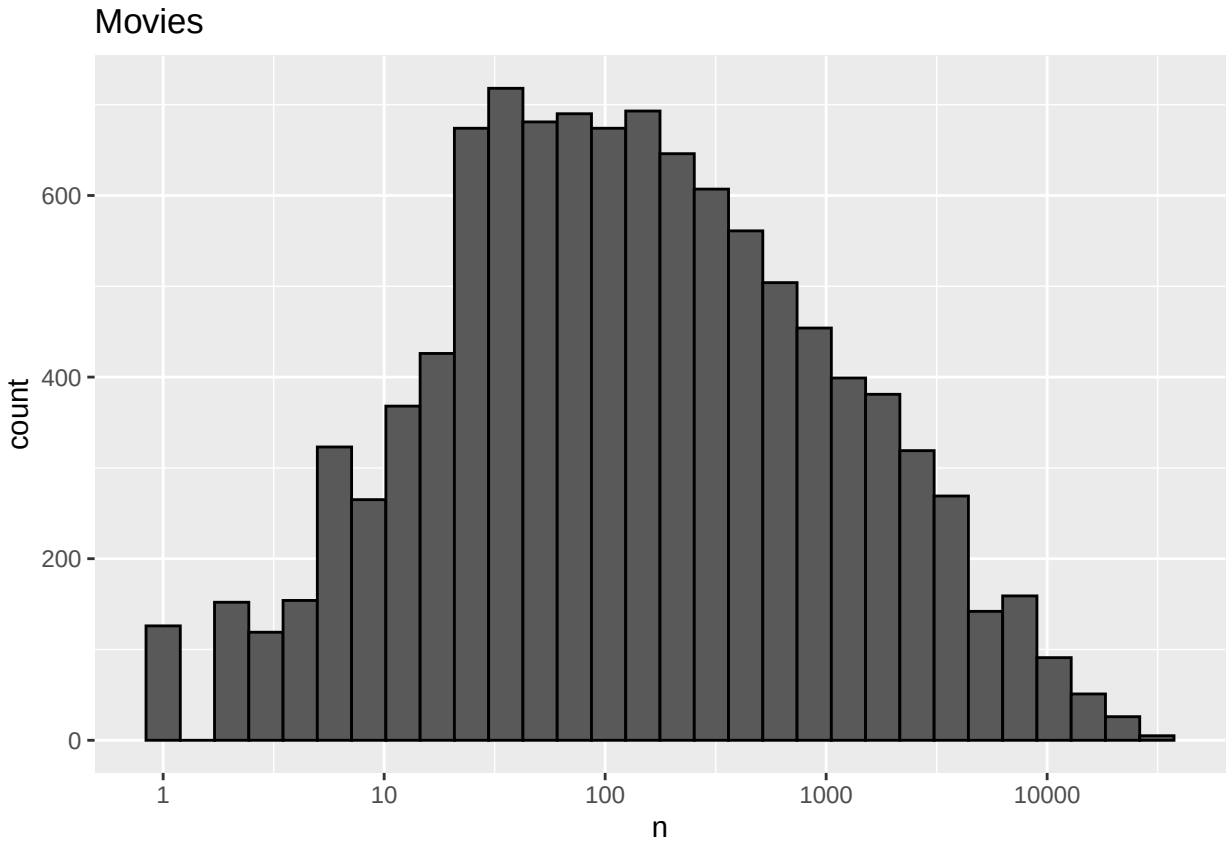
To understand the data better I reviewed the distributions of the number of ratings by user and by movie.

```
edx %>%
     dplyr::count(userId) %>%
     ggplot(aes(n)) +
     geom_histogram(bins = 30, color = "black") +
```

```
    scale_x_log10() +
    ggtitle("Users")
```




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



There is a wide distribution for each, with some users rating a lot more movies than others, and some movies being rated much more frequently than others.

Analysis

As RMSE is used for measuring success of the predictive algorithms, I start by defining a function to measure RMSE. This takes as inputs the predicted ratings when applying the algorthm in question to the `validation` dataset, and the actual ratings included in that dataset.

The function is defined as follows:

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

Average

A initial basis for measuring improvement is created by using a very basic algorithm in which every user and movie is given the same rating prediction - the average of all the ratings in the training set.

This is calculated as follows:

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

The average turns out to be 3.51 (rounded to 2d.p. for display only).

Applying the RMSE function

```
base_rmse <- RMSE(validation$rating, average_rating)
```

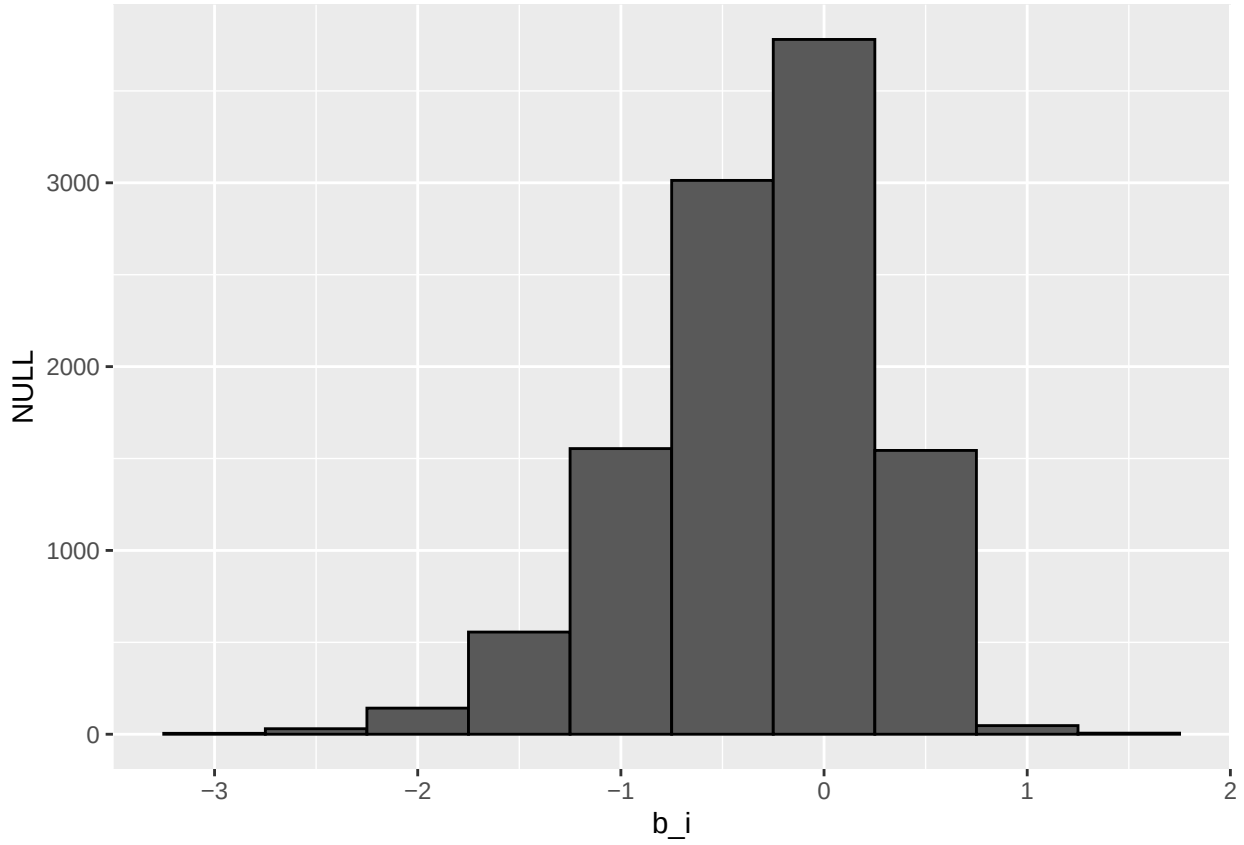
shows an RMSE of 1.0612018. An RMSE that is greater than 1, means that on average the prediction is more than one star different to the actual result.

Movie Effects

Some movies are more popular than others. I allow for the impact of this by adding a term that allows for the difference between each movie's average rating and the overall average rating (which we calculated earlier).

```
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - average_rating))
```

The differences are quite large as shown in this histogram.



The predicted ratings are calculated on the `validation` dataset using this improved algorithm, and compared to the actual ratings to get a new RMSE.

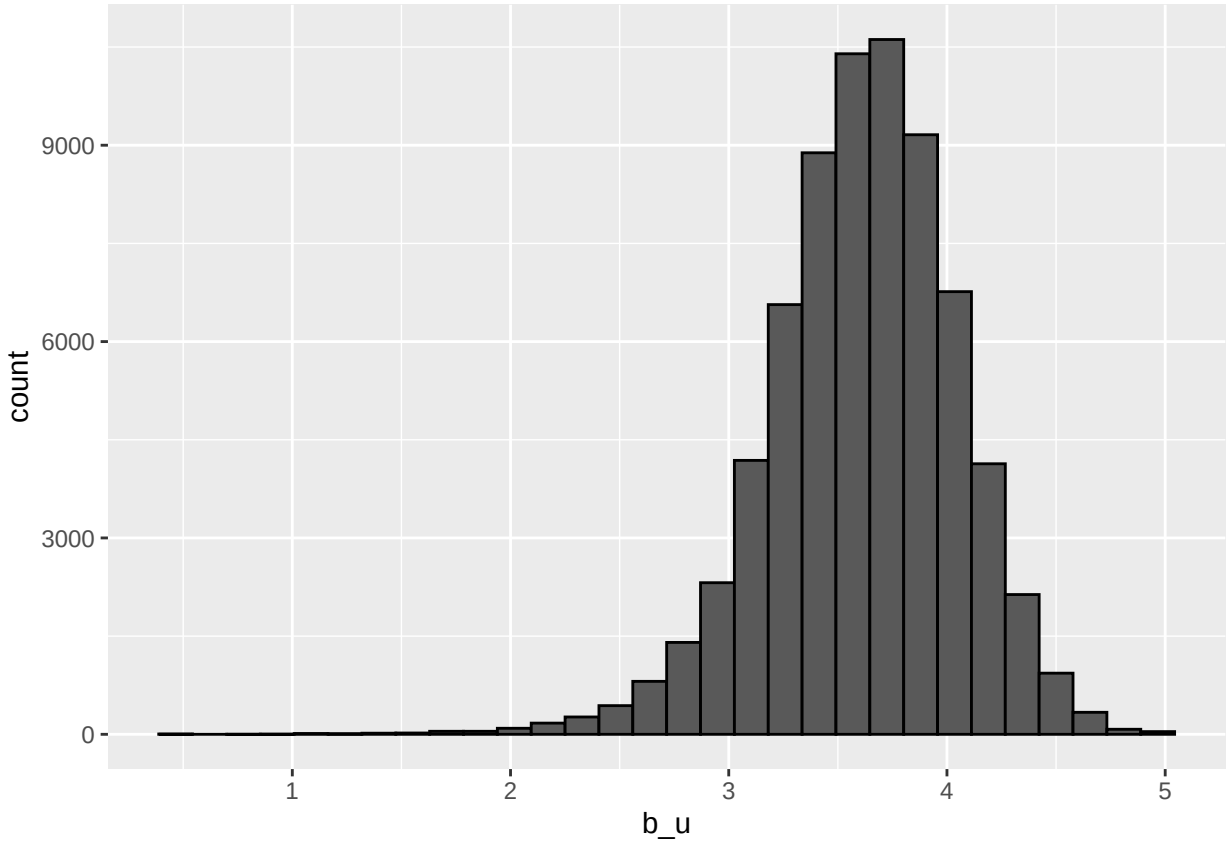
```
movie_effects <- average_rating + validation %>%
     left_join(movie_avgs, by='movieId') %>%
     .$b_i
movie_effects_rmse <- RMSE(movie_effects, validation$rating)
```

The RMSE of the algorithm with movie effects incorporated falls to 0.9439087

User Effects

Some users give harsher ratings than others. This is shown in the distribution below of user average ratings (only for those with over 100 ratings).

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



In addition to the movie effects incorporated above, I allow for this by adding a user effect term to the algorithm.

```
user_avgs <- edx %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - average_rating - b_i))

user_movie_effects <- validation %>%
     left_join(movie_avgs, by='movieId') %>%
     left_join(user_avgs, by='userId') %>%
```

```
    mutate(pred = average_rating + b_i + b_u) %>%
    .$pred

user_movie_effects_rmse <- RMSE(user_movie_effects, validation$rating)
```

With this additional user effects term incorporated the RMSE has now fallen to 0.8653488.

Regularising the User and Movie Effects

To modify the impact of large user or movie effects (i.e. large differences from the overall average) which are caused by small numbers of ratings I use regularisation. Regularisation applies a penalty to large differences caused by a small number of ratings, and uses this penalty to move the predicted difference back towards 0.

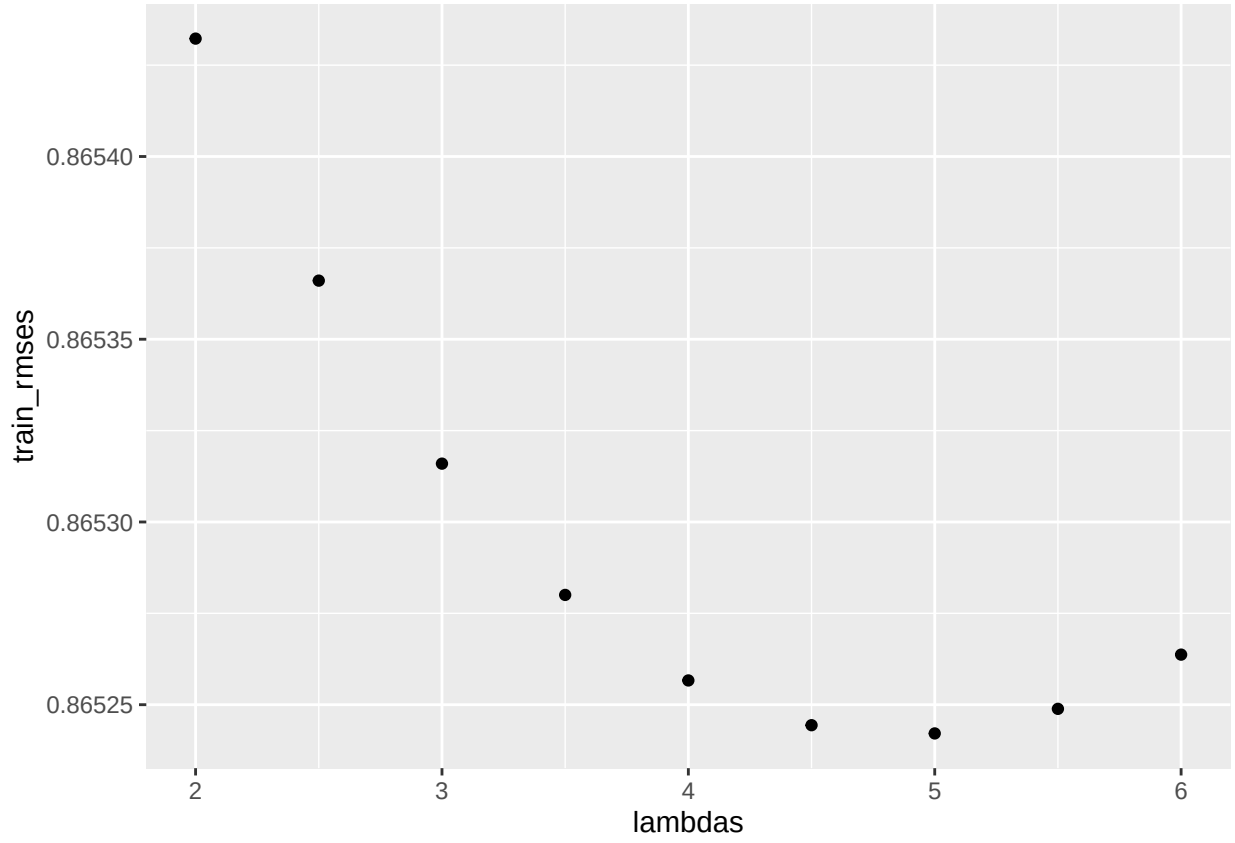
The amount by which the difference should be moved towards 0 depends on each individual dataset, and is estimated using a tuning parameter - lambda. So that I do not use the `validation` dataset to tune this parameter, I have randomly split the `edx` dataset into an 80% `train` and 20% `test` set then used these to select the lambda that minimises the RMSE on the `test` set.

```
#Splitting the edx dataset into train and test

set.seed(1, sample.kind = "Rounding")
# if using R 3.5 or earlier, use `set.seed(1)` instead

test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
train <- edx[-test_index,]
temp <- edx[test_index,]

# Make sure userId and movieId in test set are also in train set

test <- temp %>%
  semi_join(train, by = "movieId") %>%
  semi_join(train, by = "userId")

# Add rows removed from test set back into train set

removed <- anti_join(temp, test)
train <- rbind(train, removed)

# Tidy up

rm(test_index, temp, removed)

#Choose lambda

lambdas <- seq(2, 6, 0.5)

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

  mu <- mean(train$rating)

  b_i <- train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
```

```
  b_u <- train %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))

  predicted_ratings <-
    test %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred = mu + b_i + b_u) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test$rating))
})

qplot(lambdas, train_rmses)
```

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

This code produces the following plot:



and as you can see the value of `lambda` that gives the lowest RMSE is 5.

This value of `lambda` is then used to create a regularised user and movie effects algorithm on the entire `edx` dataset and then tested on the `validation` dataset as follows:

```r
# Use chosen lambda to create regularised user and movie effects model

b_i <- edx %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - average_rating)/(n()+lambda))

b_u <- edx %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - average_rating)/(n()+lambda))

#use validation to make predictions and calculate RMSE

regularised <-
  validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  mutate(pred = average_rating + b_i + b_u) %>%
  .$pred

regularised_rmse <- RMSE(regularised, validation$rating)
```

With the regularisation term added the RMSE has now fallen to 0.8648177.

Results

The RMSEs of each attempted algorithm can be summarized as follows:

Algorithm	RMSE
Base	1.0612018
Movie Effects	0.9439087
User and Movie Effects	0.8653488
Regularized	0.8648177

Conclusion

By considering the likely effects on a rating of each movie and user, it is possible to achieve a significantly greater accuracy than simply taking the average of all ratings in the database. This is further improved by regularisation - reducing the impact of users or movies with a small number of ratings. The Root Mean Squared Error (RMSE) improves from 1.0612018 for the overall average to 0.8648177 for the regularised user and movie effects model.

A number of additional pieces of work could improve the model further.

The parameter chosen for the regularisation - `lambda` was tuned using just 20% of the `edx` dataset. K-fold validation would likely improve the tuning of this parameter and may lead to a better result.

An exploration of the impact of genre, and year of release on ratings may also yield improvements to the algorithm.