HarvardX - PH125.9x Movielens project

Joe Mark Lippert

15 December 2020

Introduction

Let me make clear right from the start that the work contained herein should be credited to Prof. Rafa Irizarry and his team. Most of the code seen in the solution is found in the textbook provided to us.

Now let's begin. The dataset is the **movielens** dataset that contains ratings for movies by users. The Netflix data is not publicly available so instead we are instructed to use the **movielens** dataset. It is accessible from the Grouplens website but the code to access it, split it into 2 sets, was given to us by Professor Rafa Irizarry and is the code you see in the `setup` code chunk. The dataset has **9 000 055** rows and **6** columns. The predictor variables are:

1. **userId**,
2. **movieId**,
3. **timestamp**,
4. **title**, and
5. **genres**.

The outcome variable is **rating**.

The objective of the project is to simulate the solution offered by the winners of Netflix's 2006 challenge by applying their algorithmic approach to our project. The challenge was to improve Netflix's movie recommendation algorithm by 10%. The winners used a Machine Learning algorithm to predict the **rating** of movies by users who did not rate all the movies.

The key steps are:

- look at the data (it's always a good idea to look at the data),
- model a simple regression function,
- augment the model to incorporate the movie and user effects, and further augment the model to adjust for overfitting by incorporating the regularisation technique,
- assess the algorithm by using the RMSE (root mean squared error) loss function.

Methods/Analysis

General properties A first step in the analysis process is to look at the data. With any dataset it is always a good idea to take a look around. Identify the features and output and strategise an approach to preprocessing. Let's look at the training dataset (edx); the names of the features, output, type of vectors (variables).

```
# First let's take a glimpse at the edx (training) dataset
glimpse(edx)
```

```
## Rows: 9,000,055
## Columns: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 42...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ...
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 83...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)",...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|...
```

```
# now let's look at the first 10 items of the edx dataset
edx %>% as_tibble()
```

```
## # A tibble: 9,000,055 x 6
##    userId movieId rating timestamp title                   genres
##     <int>   <dbl>  <dbl>     <int> <chr>                   <chr>
##  1      1     122      5 838985046 Boomerang (1992)        Comedy|Romance
##  2      1     185      5 838983525 Net, The (1995)         Action|Crime|Thriller
##  3      1     292      5 838983421 Outbreak (1995)         Action|Drama|Sci-Fi|T~
##  4      1     316      5 838983392 Stargate (1994)         Action|Adventure|Sci-~
##  5      1     329      5 838983392 Star Trek: Generation~ Action|Adventure|Dram~
##  6      1     355      5 838984474 Flintstones, The (199~ Children|Comedy|Fanta~
##  7      1     356      5 838983653 Forrest Gump (1994)     Comedy|Drama|Romance|~
##  8      1     362      5 838984885 Jungle Book, The (199~ Adventure|Children|Ro~
##  9      1     364      5 838983707 Lion King, The (1994)  Adventure|Animation|C~
## 10      1     370      5 838984596 Naked Gun 33 1/3: The~ Action|Comedy
## # ... with 9,000,045 more rows
```

Each row represents a rating given by a single user for a single movie.

Now let's determine the number of unique movies, the number of unique users and the number of ratings (classes).

```
##   users movies ratings
## 1 69878  10677      10
```

Now, show the maximum possible number of ratings if each user gives a single rating to each unique movie.

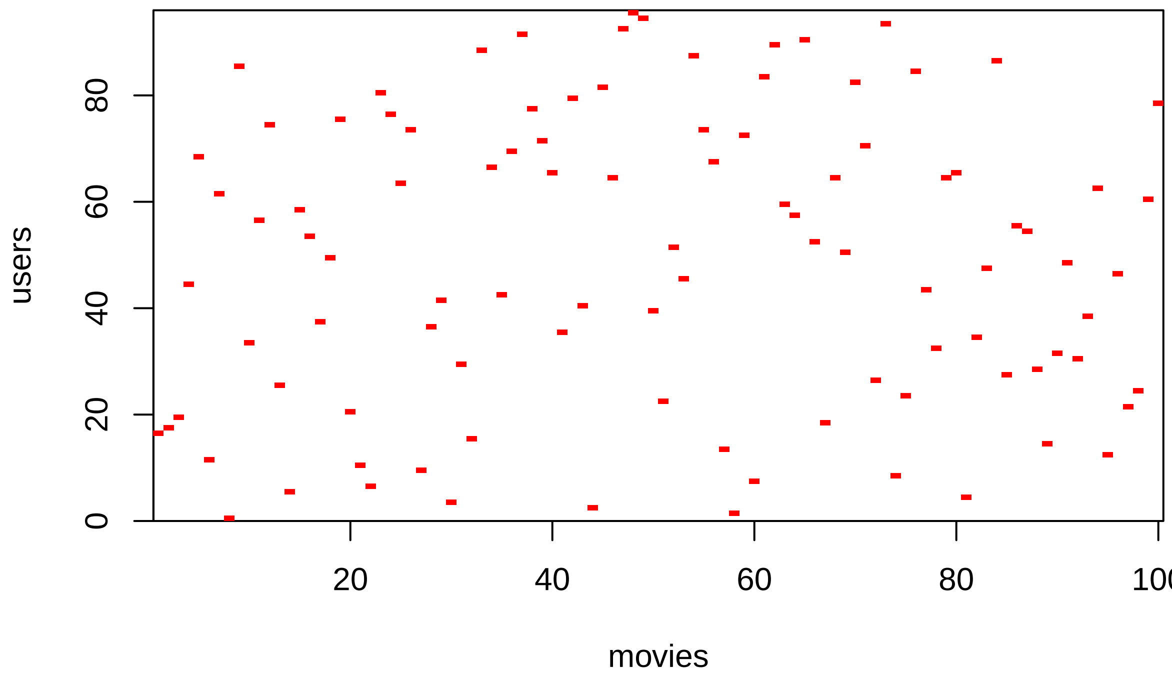
```
## [1] 746087406
```

The movielens dataset is a long dataset. It is not 'friendly' in the long format. Instead in order to see the dataset in a more useful format we need to transform it into a wide format. Not every user rated every movie. The following snippet of code shows the wide format and shows clearly that every user does not rate every movie. So we end with lots of NAs when we convert to the wide format.

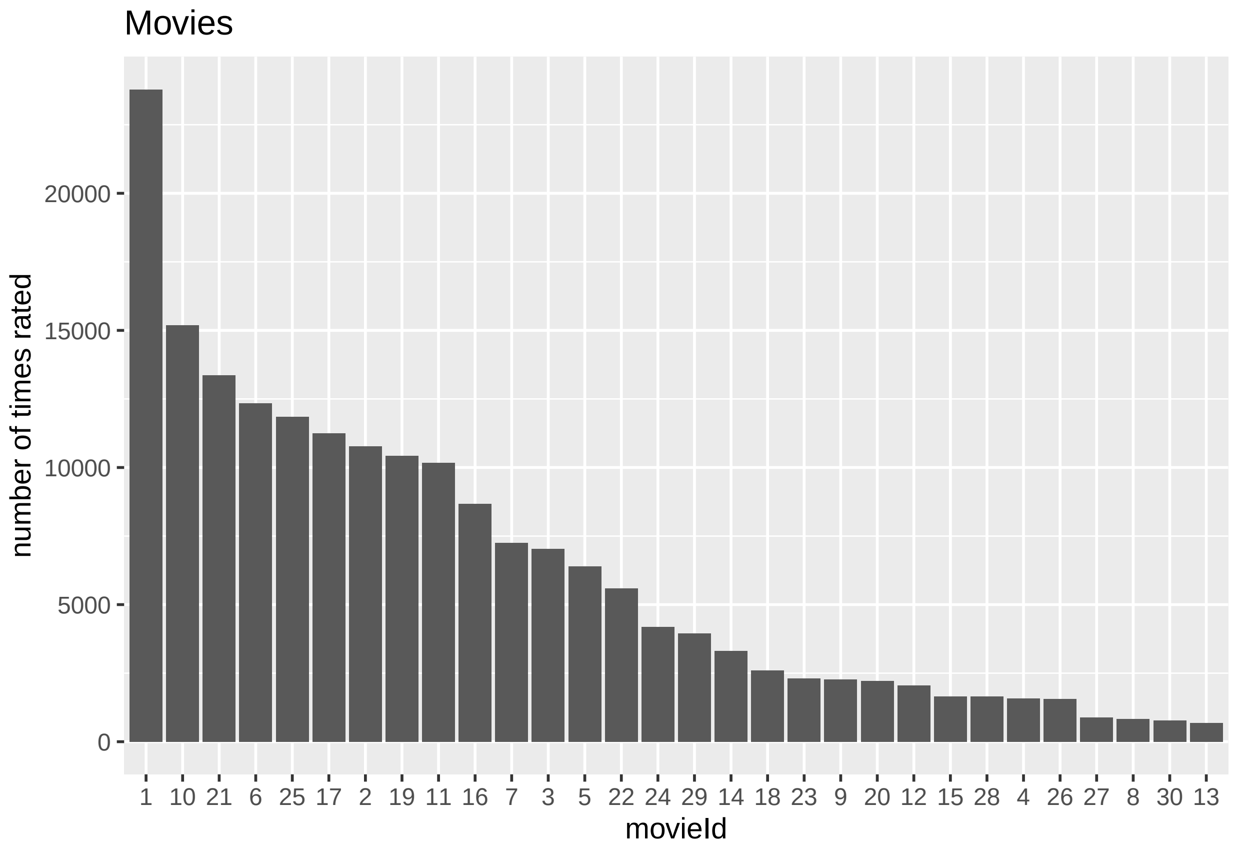
```
set.seed(2, sample.kind = "Rounding")
mat <- edx %>%
  slice_sample(n=4) %>% select(userId, title, rating) %>%
  spread(key = title, value = rating)

mat[, 2:5]
```

```
##    Crying Game, The (1992) Lethal Weapon 2 (1989) Saving Private Ryan (1998)
## 1:                      NA                     NA                          5
## 2:                      NA                     NA                         NA
## 3:                      NA                      2                         NA
## 4:                       5                     NA                         NA
##    Shrek (2001)
## 1:           NA
## 2:            3
## 3:           NA
## 4:           NA
```

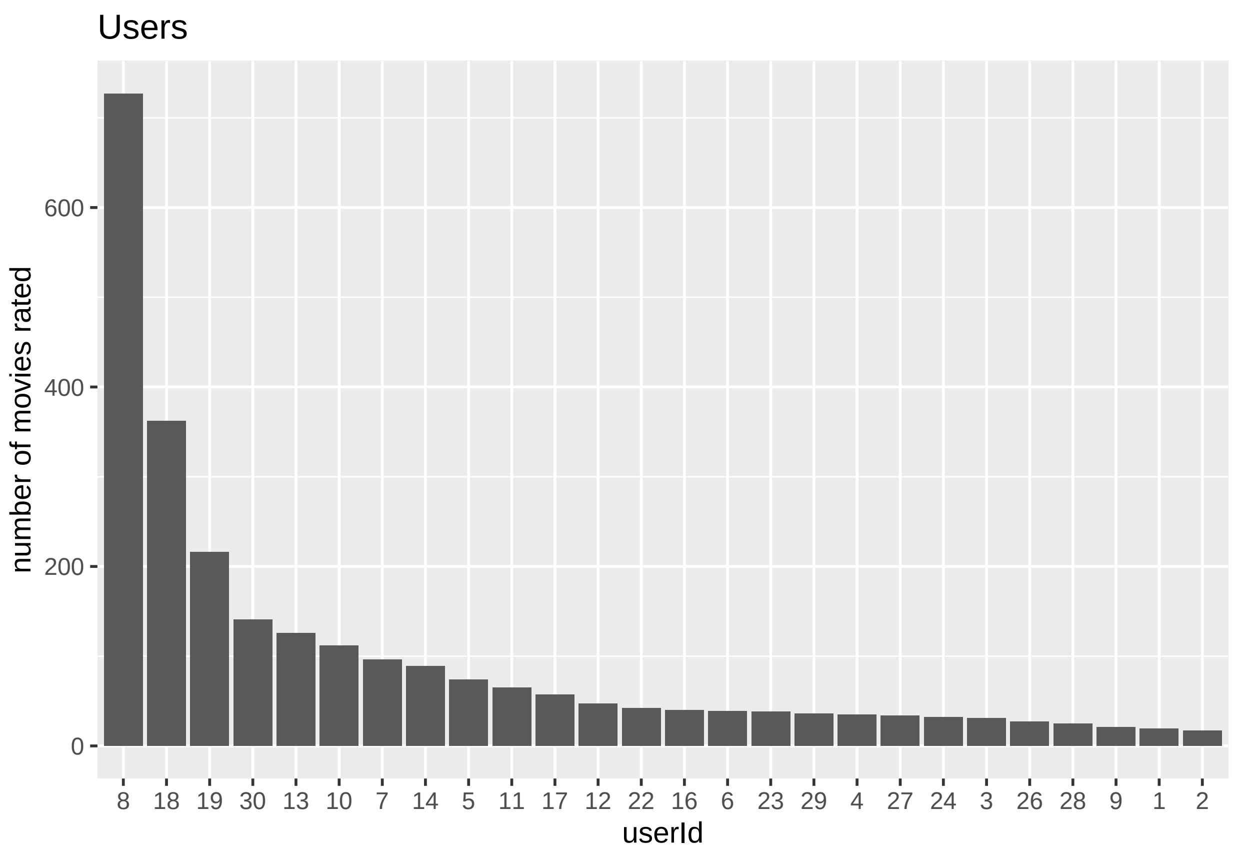
Now let's visualise a matrix with 100 movies and 100 users. Here we see every mark is a rating. The graphic shows clearly that every user does not rate every movie, hence the many gaps.



Now, let's look at some general properties of the data, like distribution of the movieId variable. The graphic below shows the varying popularity of individual movies.



Again, in the graph below we see that some users are much more engaged than others. By example, user 8 rated much more movies than user 2.



Now let's begin the process of building a predictive algorithm that predicts the rating in the spaces filled with NAs in the wide format of the data.

First, let's create a test set using edx.

```r
set.seed(755, sample.kind = "Rounding")
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2,
                                  list = FALSE)
train_set <- edx[-test_index, ]
test_set <- edx[test_index, ]
```

Let's make sure we do not include users and movies in the test set that do not appear in the train set.

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

Loss function Before we continue let us state how we shall assess and evaluate our algorithm.

The prediction error for the i^{th} data point is: $predicted\ value_i$ - $actual\ value_i$

Netflix decided that the winners would be adjudged based on the root mean squared error (RMSE) on the test set. RMSE is the standard deviation of the residuals (prediction errors). The RMSE is defined as:

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

Here is the function that computes the RMSE for vectors of ratings and their predictors:

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

Simple model Now let's begin with the simplest recommendation algorithm: A model that predicts the same rating for every movie regardless of the user.

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

$Y_{u,i}$ is the total rating per movie.

μ is the rating that we assign to each movie.

$\epsilon_{u,i}$ are the independent errors.

The estimate that minimises the RMSE is the least squares estimate of μ which is the average of all the ratings.

We calculate μ like this:

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

```
## [1] 3.512527
```

If we predict that all unknown ratings equals $\hat{\mu}$ then the RMSE is:

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

```
## [1] 1.060561
```

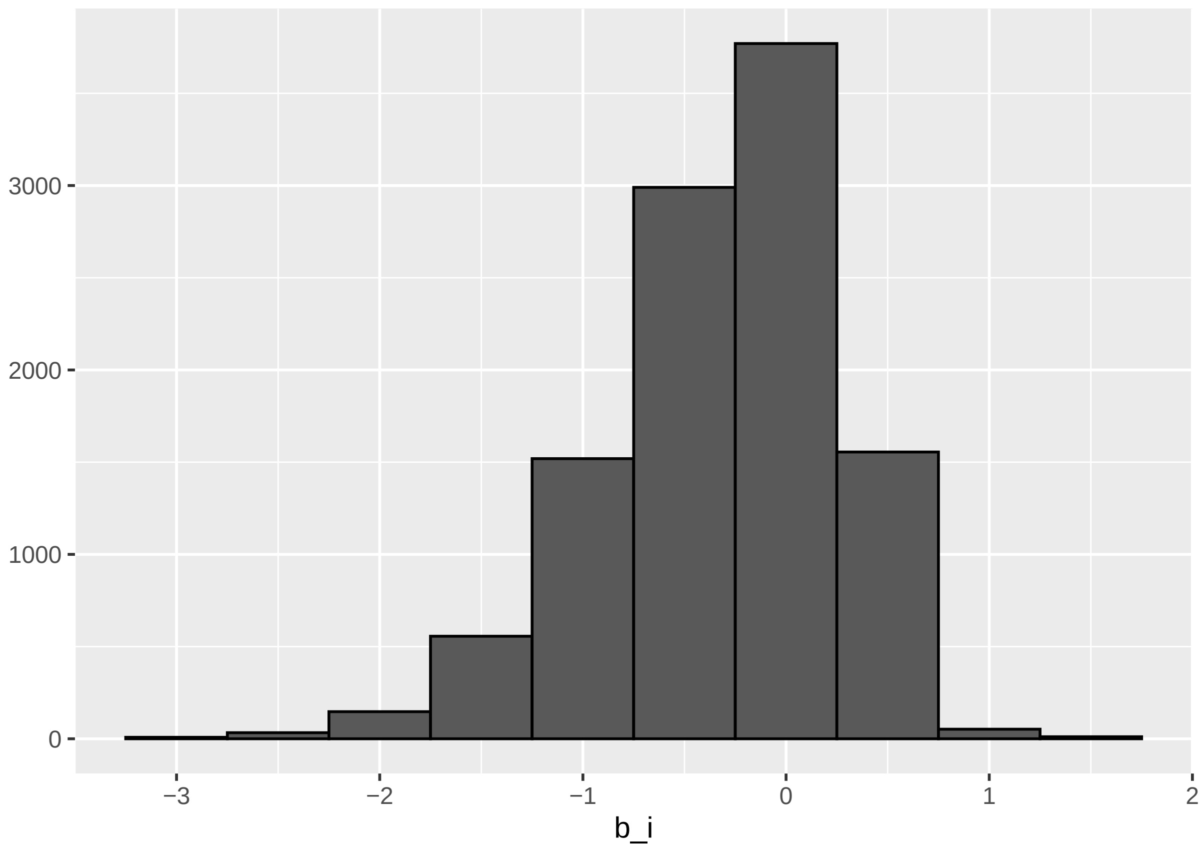
Model that incorporates movie effects Now let us include the fact that some movies are obviously more popular than others and therefore are rated differently. To reflect this fact we augment our equation to include this fact.

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

The b_i reflects the average ranking for movie i. We can use least squares to estimate b_i by using the `lm` function but because we have very many movies this process will be very slow. Instead, because we know that for this situation the least squares estimate $\hat{b_i}$ is the average of $Y_{u,i} - \hat{\mu_i}$ for each movie i. So we compute b_i as follows: (Note: going forward we drop the hat)

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

The graph below shows the variation in bias.



Now let's see how much our prediction improves when we use $\hat{y}_{u,i} = \hat{\mu}_i + \hat{b}_i$:

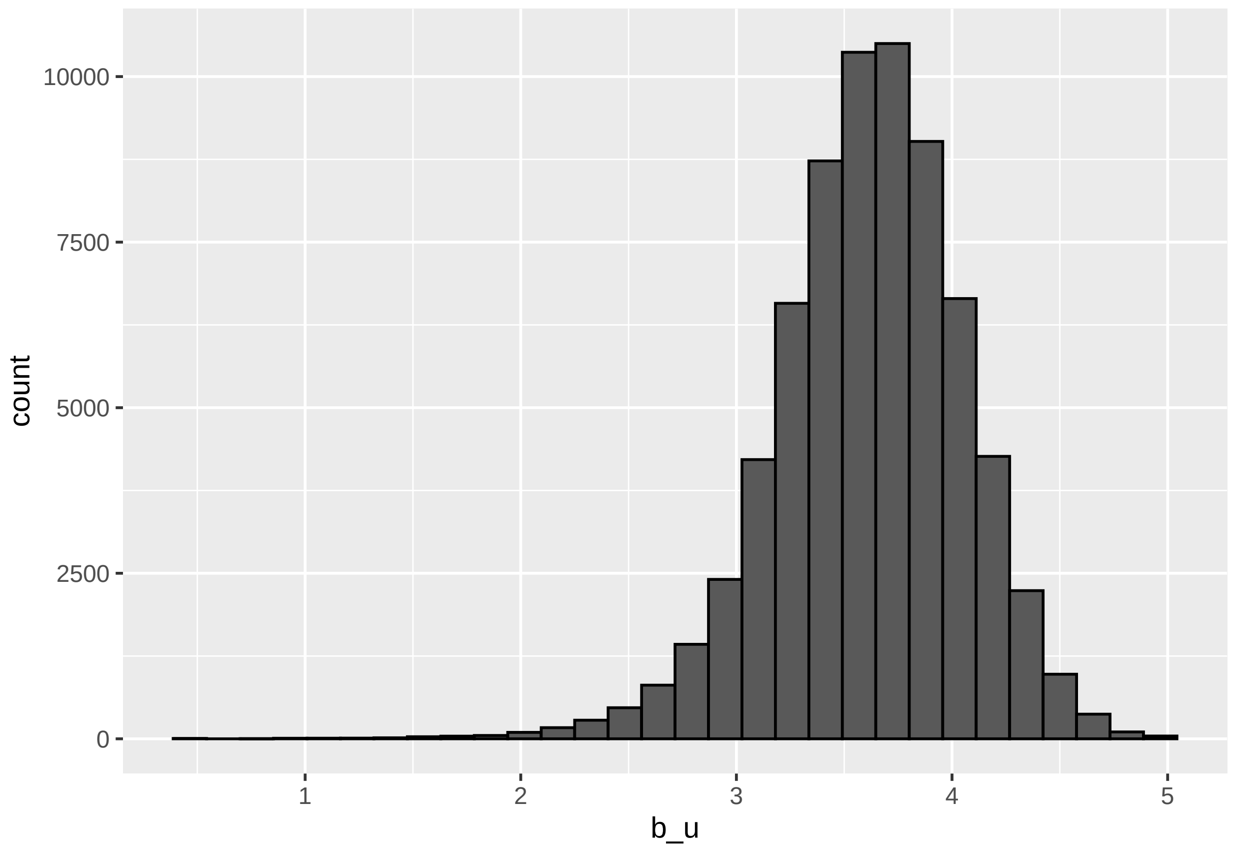
```
predicted_ratings <- mu + test_set %>%
  left_join(movie_avgs, by="movieId") %>%
  pull(b_i)

movie_effect <- RMSE(predicted_ratings, test_set$rating)
movie_effect
```

```
## [1] 0.9439868
```

We see that our loss function has improved.

User effects Let's look at the average rating of users who have rated more than 100 movies and graph the variability. Then factor this bias effect into our model.

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



From the graphic you can see that some users are harsh critics and others simply love movies and are generous in their praise. So we further augment our model to include this effect and it becomes:

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

where b_u is a user specific effect. To fit the model we could use the `lm` function but as explained above, it would be a slow process. Instead we will estimate $\hat{b}_u$ as the average of $y_{u,i} - \hat{\mu} - \hat{b}_i$.

```
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu - b_i))
# now construct predictors and see if there is an improvement to RMSE.
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)

plus_user <- RMSE(predicted_ratings, test_set$rating)
plus_user
```

```
## [1] 0.8666408
```

Regularisation Regularisation is a technique used to solve the problem of overfitting in Machine Learning models. Overfitting occurs when a model learns the detail and noise in the training data to an extent that it negatively impacts the performance of the model on new data.

How do we solve for overfitting in our model? We penalise our loss function.

Now let's look at instances of overfitting; the largest errors when we incorporate only movie effects.

```
test_set %>%
  left_join(movie_avgs, by = "movieId") %>%
  mutate(residual = rating - (mu + b_i)) %>%
  arrange(desc(abs(residual))) %>%
  slice(1:10) %>%
  pull(title)
```

```
##  [1] "Pokémon Heroes (2003)"
##  [2] "Samurai Rebellion (Jôi-uchi: Hairyô tsuma shimatsu) (1967)"
##  [3] "Shawshank Redemption, The (1994)"
##  [4] "Shawshank Redemption, The (1994)"
##  [5] "Shawshank Redemption, The (1994)"
##  [6] "Shawshank Redemption, The (1994)"
##  [7] "Shawshank Redemption, The (1994)"
##  [8] "Godfather, The (1972)"
##  [9] "Godfather, The (1972)"
## [10] "Godfather, The (1972)"
```

Now let's look at the 10 worst and 10 best movies based on only $\hat{b}_i$.

First, connect movieId to title.

```
movie_titles <- train_set %>%
  select(movieId, title) %>%
  distinct()
```

Then the 10 best movies according to our estimate $\hat{b}_i$ are:

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

```
##  [1] "Hellhounds on My Trail (1999)"
##  [2] "Who's Singin' Over There? (a.k.a. Who Sings Over There) (Ko to tamo peva) (1980)"
##  [3] "Satan's Tango (Sátántangó) (1994)"
##  [4] "Fighting Elegy (Kenka erejii) (1966)"
##  [5] "Sun Alley (Sonnenallee) (1999)"
##  [6] "Along Came Jones (1945)"
##  [7] "Angus, Thongs and Perfect Snogging (2008)"
##  [8] "Bullfighter and the Lady (1951)"
##  [9] "Blue Light, The (Das Blaue Licht) (1932)"
## [10] "Constantine's Sword (2007)"
```

And the 10 worst are:

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

```
##  [1] "Besotted (2001)"
##  [2] "Grief (1993)"
##  [3] "Altered (2006)"
##  [4] "Accused (Anklaget) (2005)"
##  [5] "Confessions of a Superhero (2007)"
##  [6] "War of the Worlds 2: The Next Wave (2008)"
##  [7] "Karla (2006)"
##  [8] "SuperBabies: Baby Geniuses 2 (2004)"
##  [9] "Disaster Movie (2008)"
## [10] "From Justin to Kelly (2003)"
```

These (10 'best' and 'worst' based upon $\hat{b}_i$ only) look obscure. See below how often they were rated.

```
train_set %>% count(movieId) %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(desc(b_i)) %>%
  slice(1:10) %>%
  pull(n)
```

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

```
train_set %>% count(movieId) %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(b_i) %>%
  slice(1:10) %>%
  pull(n)
```

```
##  [1]   2   1   1   1   1   2   2  44  27 159
```

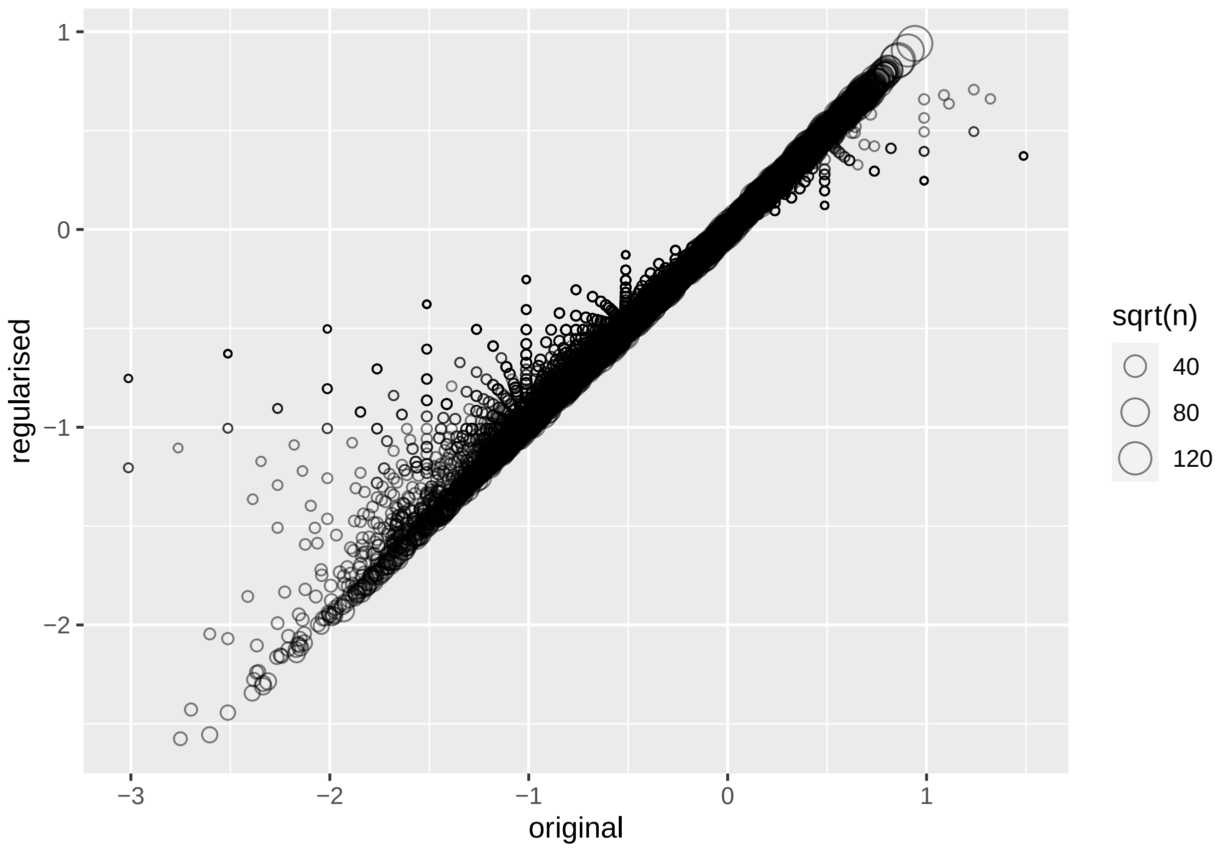
The 'best' and 'worst' movies were mostly rated by few users. This represents noise and uncertainty and can result in large errors and consequently an increase in RMSE.

Now to remedy these inconsistencies, we use regularisation to ameliorate the effects of different levels of uncertainty.

```

lambda <- 3
mu <- mean(train_set$rating)
movie_reg_avgs <- train_set %>%
  group_by(movieId) %>%
  summarise(b_i = sum(rating - mu)/(n() + lambda), n_i = n())
```

Regularisation for movie effects To see how the estimates shrink, below see a plot of the regularised estimates versus the least squares estimates.

```
tibble(original = movie_avgs$b_i, regularised = movie_reg_avgs$b_i,
       n = movie_reg_avgs$n_i) %>%
  ggplot(aes(original, regularised, size = sqrt(n))) +
  geom_point(shape = 1, alpha = 0.5)
```



Now let's look at the top 10 movies based on the penalised estimates $\hat{b}_i(\lambda)$.

```
train_set %>%
  count(movieId) %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(desc(b_i)) %>%
  slice(1:10) %>%
  pull(title)
```

```
##  [1] "Shawshank Redemption, The (1994)"
##  [2] "Godfather, The (1972)"
##  [3] "Schindler's List (1993)"
##  [4] "Usual Suspects, The (1995)"
##  [5] "Rear Window (1954)"
##  [6] "Casablanca (1942)"
##  [7] "Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)"
##  [8] "Double Indemnity (1944)"
##  [9] "Godfather: Part II, The (1974)"
## [10] "Seven Samurai (Shichinin no samurai) (1954)"
```

And here are the 10 'worst' movies.

```
train_set %>%
  count(movieId) %>%
  left_join(movie_reg_avgs, by = "movieId") %>%
  left_join(movie_titles, by = "movieId") %>%
  arrange(b_i) %>%
  slice(1:10) %>%
  pull(title)
```

```
##  [1] "SuperBabies: Baby Geniuses 2 (2004)"
##  [2] "From Justin to Kelly (2003)"
##  [3] "Pokémon Heroes (2003)"
##  [4] "Disaster Movie (2008)"
##  [5] "Pokemon 4 Ever (a.k.a. Pokémon 4: The Movie) (2002)"
##  [6] "Glitter (2001)"
##  [7] "Barney's Great Adventure (1998)"
##  [8] "Gigli (2003)"
##  [9] "Yu-Gi-Oh! (2004)"
## [10] "Faces of Death: Fact or Fiction? (1999)"
```

Does the penalised version improve our result?

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

reg_movie <- RMSE(predicted_ratings, test_set$rating)
reg_movie
```

```
## [1] 0.9439252
```

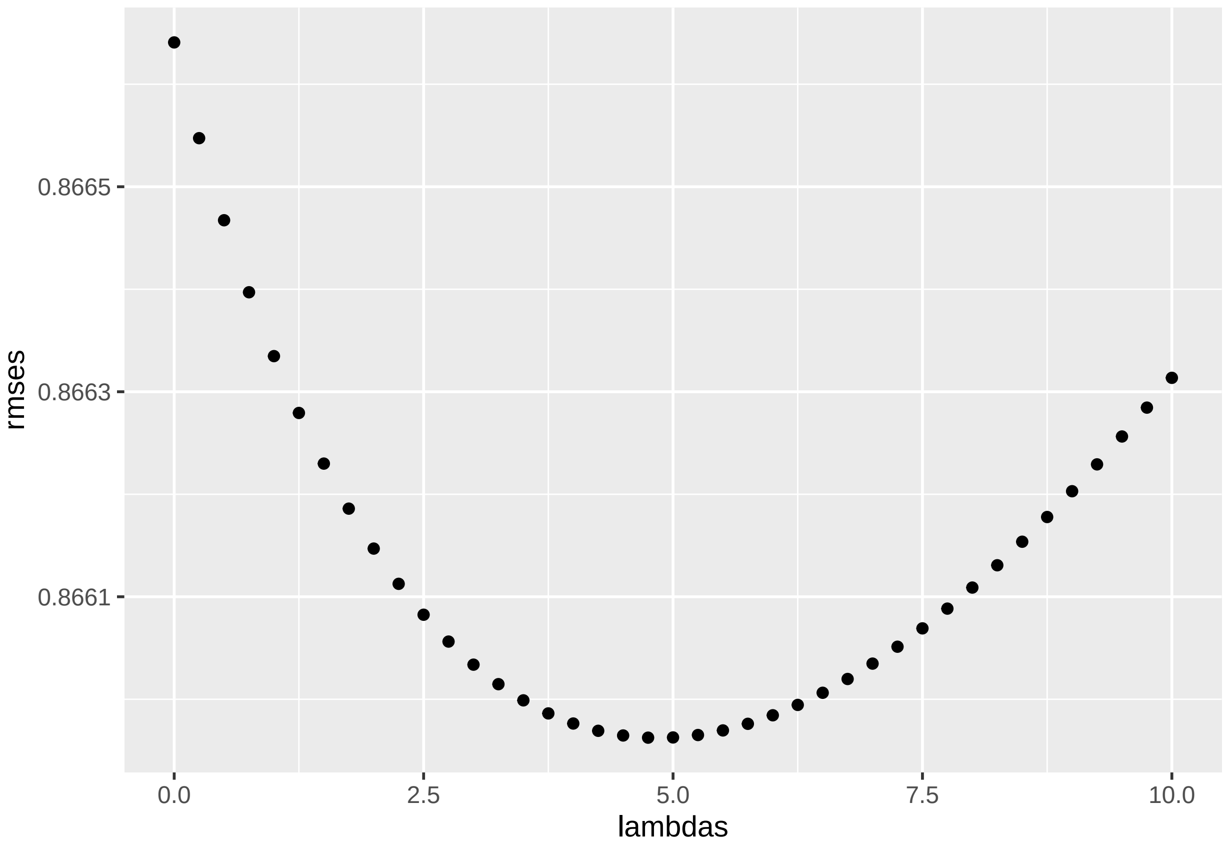
Regularisation for movie + user effects (cross-validation) We use regularisation to estimate for user effects as well. Below is the equation that describes the estimate.

$\frac{1}{N}\sum_{u,i}\left(\hat{y}_{u,i}-\mu-b_i-b_u\right)^2+\lambda\left(\sum_i b_i^2+\sum_u b_u^2\right)$

The solution (below) shows us using cross-validation to pick the parameter λ.

```
lambdas <- seq(0, 10, 0.25)

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

  mu <- mean(train_set$rating)

  b_i <- train_set %>%
    group_by(movieId) %>%
    summarise(b_i = sum(rating - mu)/(n() + l))

  b_u <- train_set %>%
    left_join(b_i, by = "movieId") %>%
    group_by(userId) %>%
    summarise(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) %>%
    pull(pred)

  return(RMSE(predicted_ratings, test_set$rating))
})
```

```r
qplot(lambdas, rmses)
```



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

```
## [1] 4.75
```

```r
reg_movie_user <- min(rmses)
reg_movie_user
```

```
## [1] 0.8659626
```

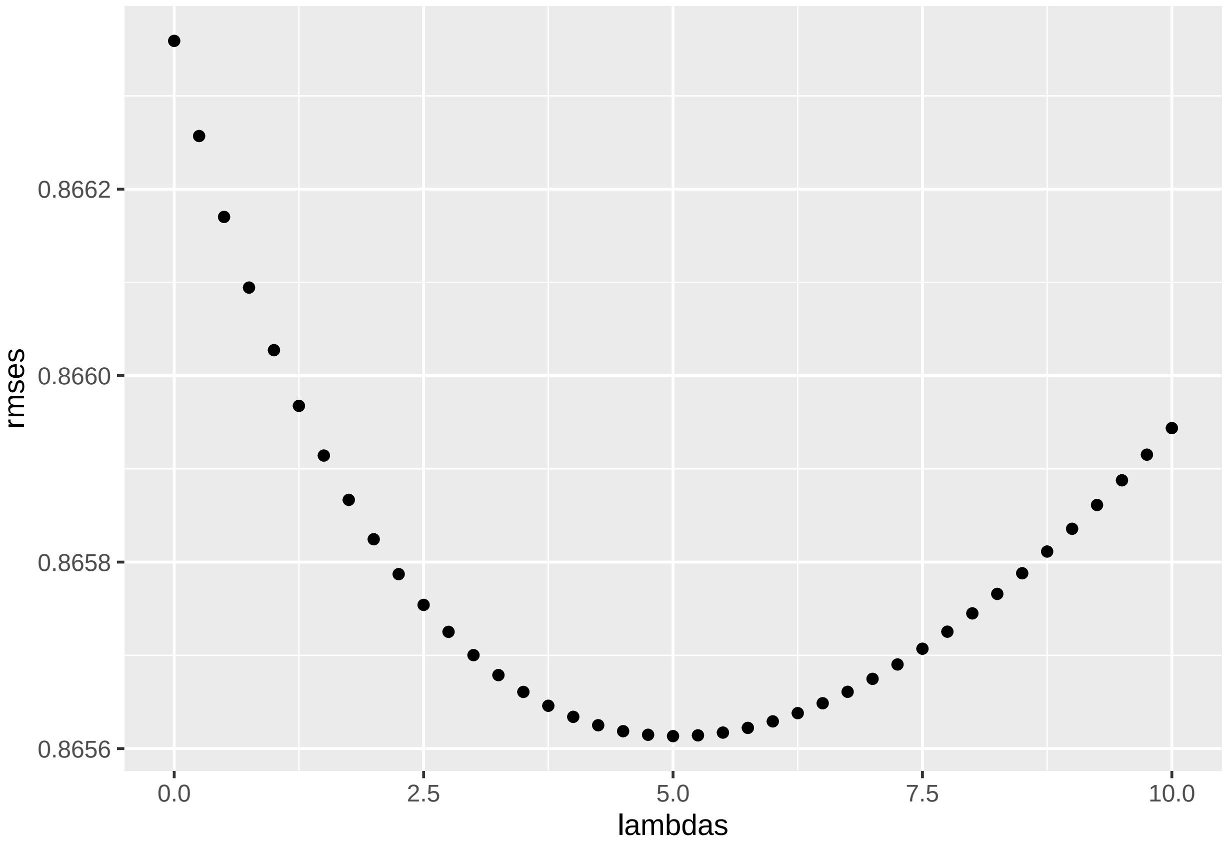
Results

Results with test set The results you see here mostly simulates the results seen in the course literature. Having said this, some of the code in the text contained some inconsistencies that required one to decipher these in order to complete the task. This task required a considerable amount of energy from a mind as limited in knowledge such as I have.

```
## # A tibble: 5 x 2
##   method                                RMSE
##   <chr>                                <dbl>
## 1 Just the average                     1.06
## 2 Movie Effect Model                   0.944
## 3 Movie + User Effects Model           0.867
## 4 Regularised Movie Effect Model       0.944
## 5 Regularised Movie + User Effect Model 0.866
```

Results with validation set

```
## [1] 1.061196

## [1] 0.9440559

## [1] 0.866359

## [1] 0.9439762
```



```
## [1] 5

## [1] 0.8656134

## # A tibble: 5 x 2
##   method                               RMSE
##   <chr>                               <dbl>
## 1 Just the average                    1.06
## 2 Movie Effect Model                  0.944
## 3 Movie + User Effects Model          0.866
## 4 Regularised Movie Effect Model      0.944
## 5 Regularised Movie + User Effect Model 0.866
```

Conclusion

This report articulates the primary objective of 1 of the 2 capstone projects; to simulate the solution offered by the winners of Netflix's 2006 challenge by applying their algorithmic approach to our project. We are called upon to predict the ratings of movies in a dataset by users, where no ratings were given.

The limitations evident in this report are connected to the level of understanding the author has about Machine Learning, particularly his lack of depth of knowledge required to fully comprehend the many facets of ML. In the absence of a solid foundation in complementary subjects, such as linear algebra and principal components analysis, to name a few, it is extremely challenging to tackle a projectas complex as this one.

Typically datasets have variables populated with data. NAs are the exception and strategies are available to tackle this obstacle. In this assignment, we have a dataset where most of the variables are populated with NAs. This makes this assignment an extremely difficult task.

The winners of the competition won $1 million. I hypothesise that the reason for the generous sum can be found in recognition of the complexity of the challenge.

I recommend that Harvard rethink placing this assignment before persons until they make clear that course registrants should be well versed in the disciplines listed above.

In conclusion, I am curious to see how this challenge can be addressed using decision trees or random forests.