MovieLens: Predicting Movie Ratings with Machine Learning

Simon Piper

20/04/2020

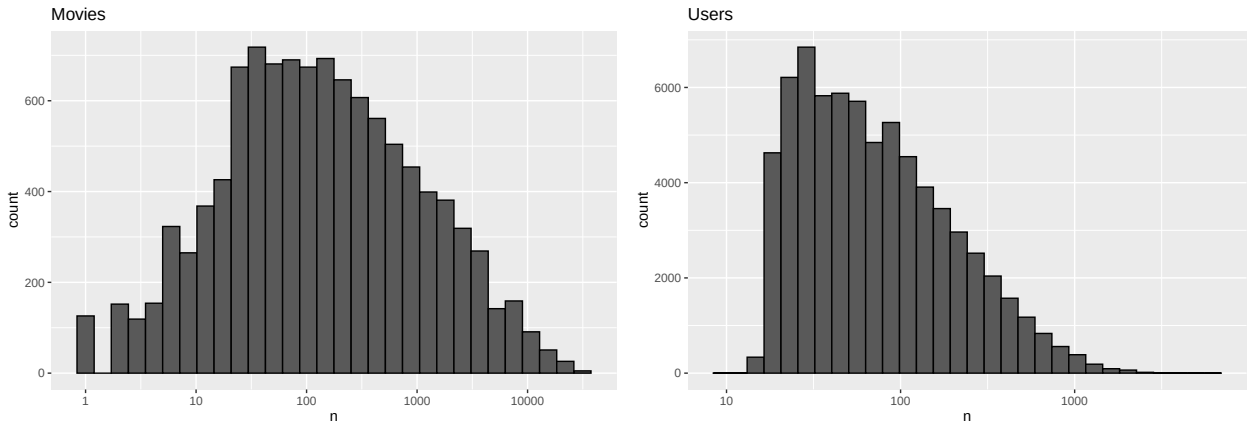
Overview

Many different businesses employ a system which allows their customers to rate their products, which in turn can be used to generate predictions of ratings and then recommend similar products which they believe that particular user would have rated highly. Netflix is one company which uses such a system, predicting how many stars a user may give, with one being low and five describing a great movie.

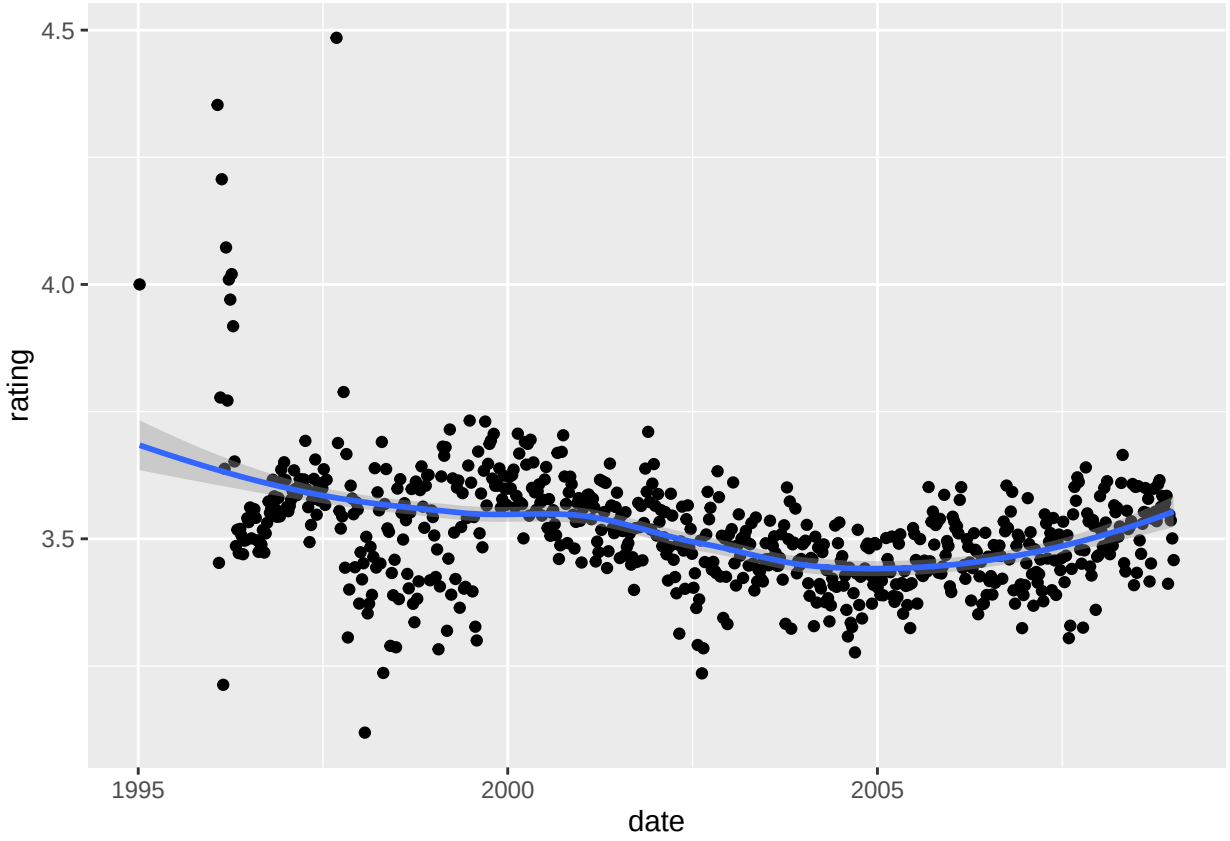
In this report I create a movie recommendation system using a dataset containing over 10 million ratings of 10,000 movies, given by 72,000 users between 1995 and 2009. This is then split into two, with the dataset used to train my algorithmn containing ~9 million observations, and the other final valiation dataset ~1 million observations. The below table shows the structure of the dataset I use for training, displaying how each row represents one rating given by a unique user for a specific film:

```
## # A tibble: 9,000,055 x 7
##    userId movieId rating timestamp title            genres            date
##     <int>   <dbl>  <dbl>     <int> <chr>            <chr>             <date>
##  1      1     122      5 838985046 Boomerang (1992) Comedy|Romance    1996-08-02
##  2      1     185      5 838983525 Net, The (1995)  Action|Crime|Thr~ 1996-08-02
##  3      1     292      5 838983421 Outbreak (1995)  Action|Drama|Sci~ 1996-08-02
##  4      1     316      5 838983392 Stargate (1994)  Action|Adventure~ 1996-08-02
##  5      1     329      5 838983392 Star Trek: Gene~ Action|Adventure~ 1996-08-02
##  6      1     355      5 838984474 Flintstones, Th~ Children|Comedy|~ 1996-08-02
##  7      1     356      5 838983653 Forrest Gump (1~ Comedy|Drama|Rom~ 1996-08-02
##  8      1     362      5 838984885 Jungle Book, Th~ Adventure|Childr~ 1996-08-02
##  9      1     364      5 838983707 Lion King, The ~ Adventure|Animat~ 1996-08-02
## 10      1     370      5 838984596 Naked Gun 33 1/~ Action|Comedy     1996-08-02
## # ... with 9,000,045 more rows
```

Due to the fact that not every user has rated every film in the dataset, there is an added complexity, as each outcome Y will have a different set of predictors. The distributions below show how some movies receive a lot of ratings, such as blockbuster films, while others very few, perhaps such as less popular independant films. Furthermore, some users are much more active at rating movies than others.



Plotting the average film rating against time does not show strong evidence of a time trend throughout the dataset, or support inclusion of such a trend within the analysis:



The final accuracy of the algorithmn developed will be measured using a loss function; the residual mean squared error (RMSE). This is similar to the standard deviation or typical error when predicting a movie rating. Where $y_{u,i}$ is the rating for movie i by user u, in predicting $\hat{y}_{u,i}$, and with N being the number of user/movie combinations, the RMSE can be defined as:

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

Method & Analysis

In order to develop the algorithmn and carry out the analyis, the larger training dataset is itself split into a training set and a test set. The data is partitioned in such a way to ensure that movies and users which don't appear in the training set, are not included in the test set.

```
library(dplyr)
library(caret)
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,]
test_set <- test_set %>% semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

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

Each model is trained using the training dataset and then tested using the test dataset. After parameters are selected and model tuned, the original dataset is then used to train the final model against the validation dataset.

Model 1: Just the Average The simplest model to predict movie ratings is one which predicts the same rating for all movies regardless of user, and where any differences are explained by random variation. This can be shown by the following:

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

where $\epsilon_{u,i}$ is the independent random error with a distribution centred at 0 and μ the "true" rating. The estimate of μ which minimises the chosen loss function is the average of all film ratings:

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

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

Using this estimate yields the following RMSE:

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

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

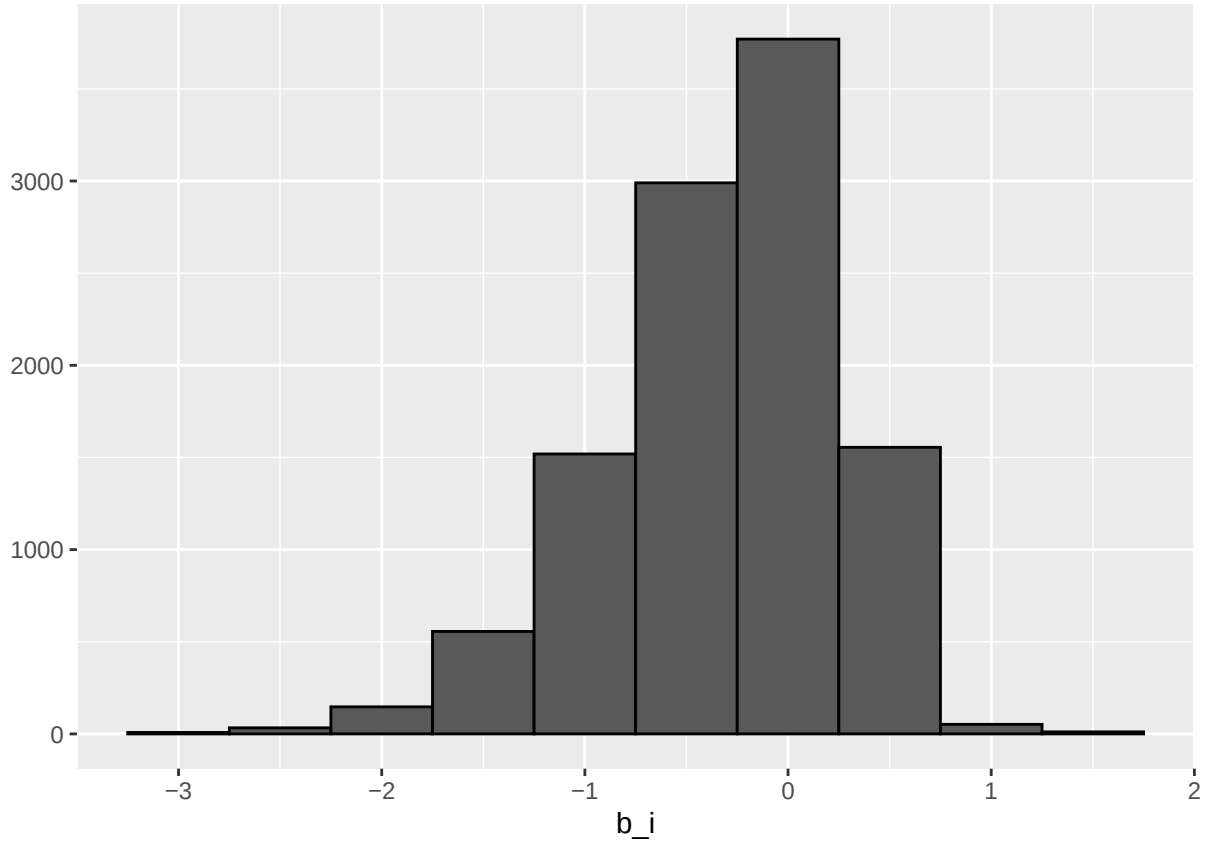
Model 2: Movie Effects Because some movies are just generally rated higher than others, Model 1 can be augmented by adding the effect or bias of each movie in the dataset:

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

where b_i represents the average rating for movie i.

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

The following plot shows the large degree of variation for these estimates of b_i:



The RMSE can then be calculated:

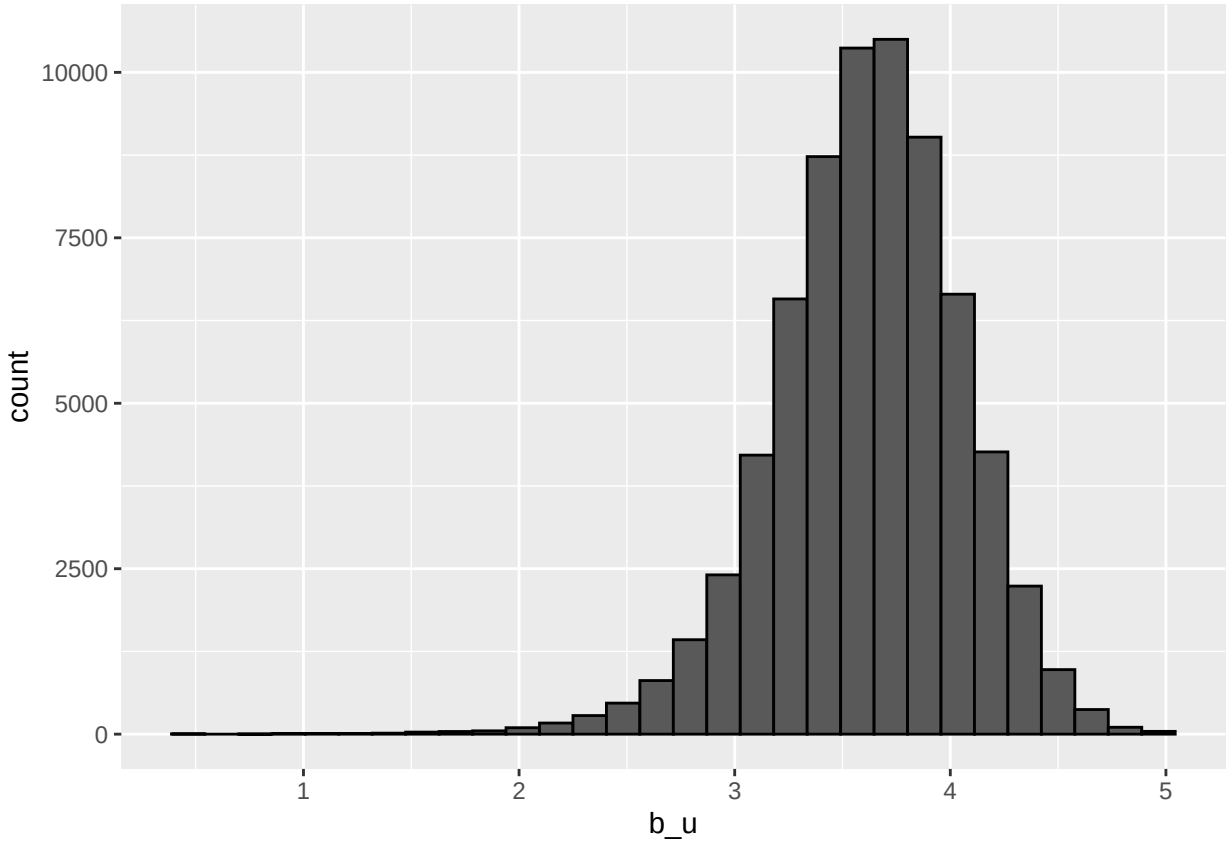
```
predicted_ratings <- mu + test_set %>% left_join(movie_avgs, by = "movieId") %>%
  pull(b_i)
movie_rmse <- RMSE(predicted_ratings, test_set$rating)
movie_rmse
```

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

Model 3: Movie & User Effects Model 2 can be further extended by adding b_u; a user-specific effect. This is to capture the effect, or bias, of some users being perhaps characteristically harsher critics, while other users may enjoy every movie. This can be written as:

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

Examining the average ratings for users who rated over 100 films shows significant variability, and supports inclusion of the user-specific effect b_u:



In a similar fashion to the previous model, $\hat{b}_u$ can be estimated as the average of $\hat{y}_{u,i} - \hat{\mu} - \hat{b}_i$:

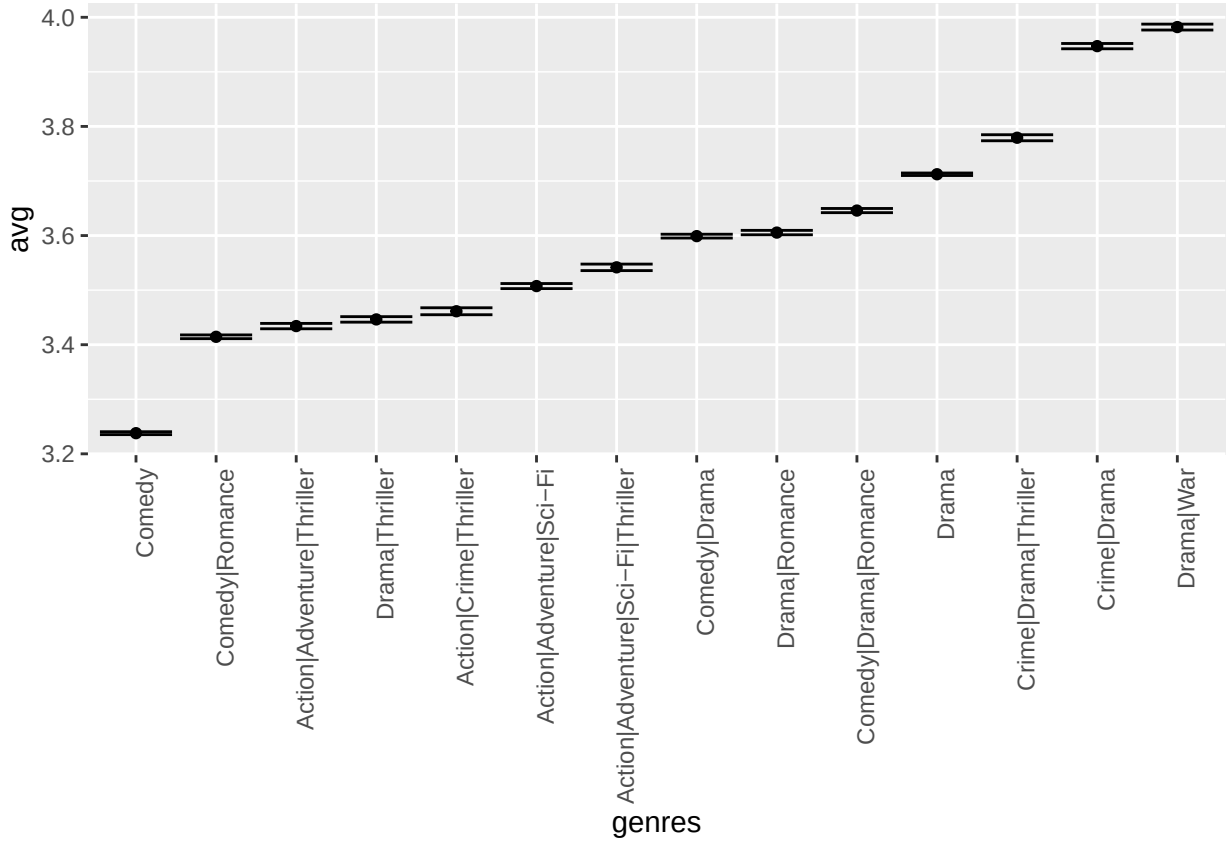
```
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - mu - b_i))

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

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

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

Model 4: Movie, User & Genre Effects In order to capture the possilibity of a genre-effect, or bias, the previous model is extended. Some movies fall under a number of different genres, while others only a few, and movies associated with certain genres may generally achieve higher ratings. To allow for this the following model can be estimated:

$$Y_{u,i} = \mu + b_i + b_u + \sum_{k=1}^{K} x_{u,i}^k \beta_k + \epsilon_{u,i}, \quad with\ x_{u,i}^k = 1\ if\ g_{u,i}\ is\ genre\ k$$

Plotting genre combinations of those that gained over 100,000 ratings against average rating shows significant variation and supports inclusion:




```
genres_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by="userId") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - mu - b_i - b_u))

predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genres_avgs, by="genres") %>%
  mutate(pred = mu + b_i + b_u + b_g) %>%
  pull(pred)

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

```
## [1] 0.8662908
```

Model 5: Regularised Movie Effects Because some movies may only have a very small number of ratings, this can result in noisy estimates of $\hat{b}_i$ when drawing from a small sample, and also give rise to over-training of the model. Therefore regularisation can be used to introduce a penalty term in order to penalise large estimates derived from small samples and constrain total effect sizes. Estimates of $\hat{b}_i$ can now be achieved by minimising the following equation:

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i)^2+\lambda\sum_i b_i^2$$

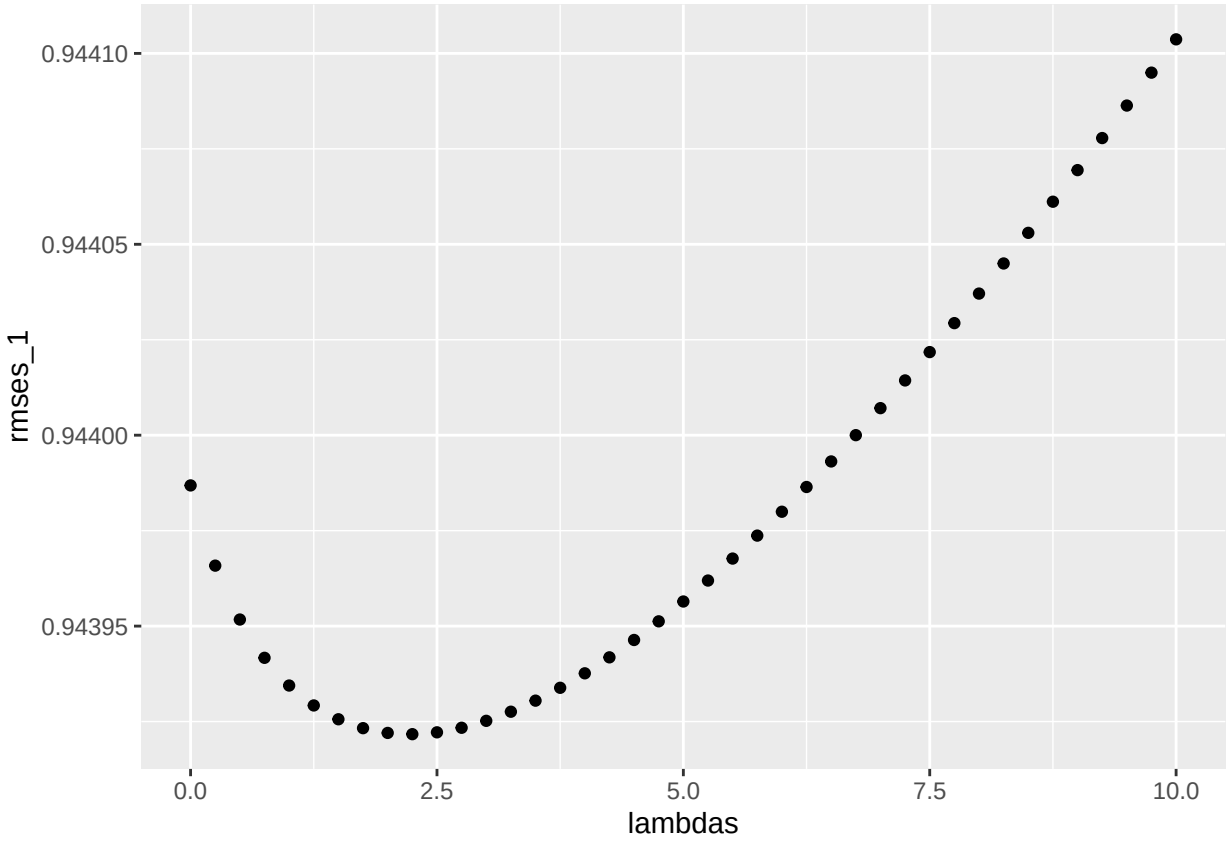
Where the first term is the least squares and the second a penalty term which gets larger when many b's are large. The values of b that minimise this equation are given by:

$$\hat{b}_i(\lambda)=\frac{1}{\lambda+n_i}\sum_{u=1}^{n_i}(Y_{u,i}-\hat{\mu})$$

where n_i is a number of ratings for movie i. Therefore λ can be considered a tuning parameter. In the case that n_i is large, λ is effectively ignored, and when n_i is small, the estimate b_i is shrunk towards zero. λ can be selected through cross-validation.

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

rmses_1 <- sapply(lambdas, function(l){

  mu <- mean(train_set$rating)

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

  predicted_ratings <- test_set %>%
    left_join(b_i, by='movieId') %>%
    mutate(pred = mu + b_i) %>%
    pull(pred)
  return(RMSE(predicted_ratings, test_set$rating))
})
```

This plot shows the optimal tuning parameter, lambda, which minimises the RMSE:




```
## [1] 2.25
```

```
min(rmses_1)
```

```
## [1] 0.9439217
```

Regularisation appears to have introduced a small improvement in RMSE compared to Model 2.

Model 6: Regularised Movie & User Effects The previous model can be augmented by also performing regularisation on the user-specific parameter, b_u. This allows for estimates to be achieved through minimising the following equation:

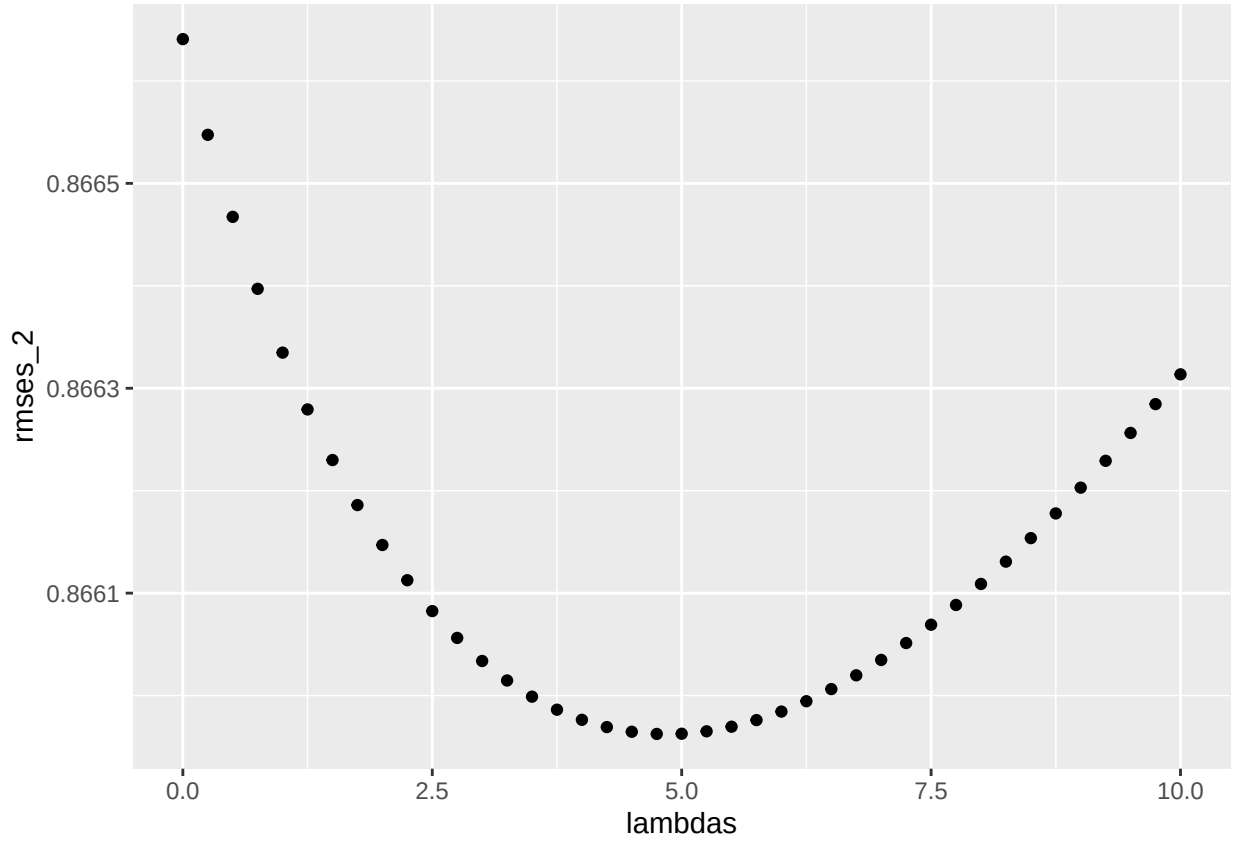
$$\frac{1}{N}\sum_{u,i}(y_{u,i} - \mu - b_i - b_u)^2 + \lambda(\sum_i b_i^2 + \sum_u b_u^2)$$

Again, cross-validation can be performed to select the optimal lambda, λ, which minimises the chosen loss function:

```
lambdas <- seq(0, 10, 0.25)
rmses_2 <- 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) %>%
    pull(pred)

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

The plot below displays the optimal value of λ:




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

```
min(rmses_2)
```

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

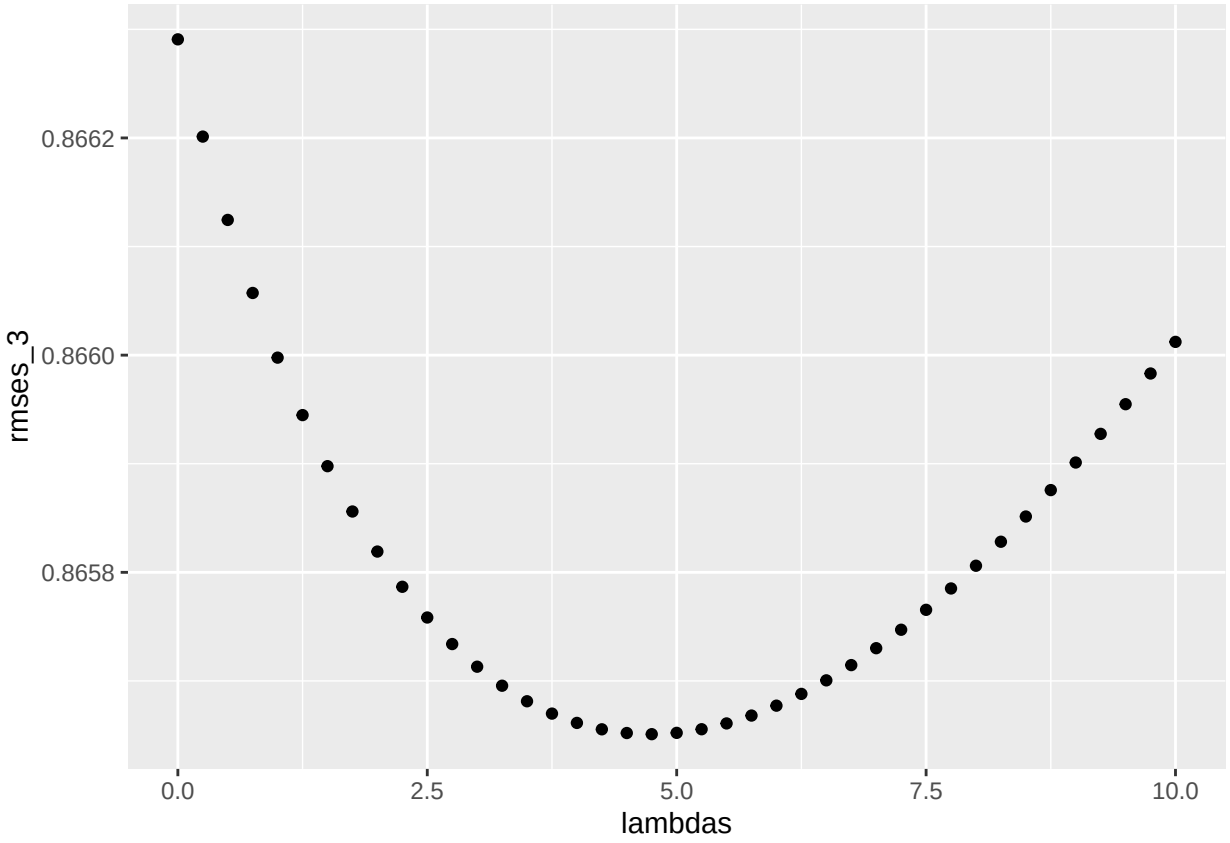
Evidently, extending regularisation to the user-specific effect results in a further decrease in RMSE.

Model 7: Regularised Movie, User & Genre Effects As there is also variation in the number of ratings across different genre categories, regularisation is also applied to the genre effect. Estimation of model parameters can be achieved through minimising the following:

$$\frac{1}{N}\sum_{u,i}(y_{u,i}-\mu-b_i-b_u-\sum_{k=1}^{K}x_{u,i}^{k}\beta_k)^2+\lambda(\sum_i b_i^2+\sum_u b_u^2+(\sum_{k=1}^{K}x_{u,i}^{k}\beta_k)^2), \quad with\ x_{u,i}^{k}=1\ if\ g_{u,i}\ is\ genre\ k$$

```
lambdas <- seq(0, 10, 0.25)
rmses_3 <- 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))

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

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

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

Cross-validation of the training dataset yields the following value of λ:




```
lambda <- lambdas[which.min(rmses_3)]
lambda
```

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

```
min(rmses_3)
```

```
## [1] 0.865651
```

This model produces the greatest improvement in RMSE relative to the other models tested.

Final Model: Regularised Movie, User & Genre Effects (Validation Dataset) Now that parameters for inclusion have been selected, as well as the corresponding tuning parameter, λ, the final model can now be evaluated. This is performed using the entire inital non-partitioned dataset as a training dataset, against the unused validation dataset as the testing dataset.

```
l <- 4.75
mu <- mean(edx$rating)

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

b_u <- edx %>%
```

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

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

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

rmses_f <- RMSE(predicted_ratings, validation$rating)

min(rmses_f)
```

```
## [1] 0.8644514
```

This final model achieves an RMSE of 0.8644514.

Results

The below table displays a summary of the results with corresponding RMSE for each of the different models tested:

Model	RMSE
Just the average	1.0605613
Movie Effect	0.9439868
Movie + User Effect	0.8666408
Movie + User + Genre Effect	0.8662908
Regularised Movie Effect	0.9439217
Regularised Movie + User Effect	0.8659626
Regularised Movie + User + Genre Effect	0.8656510
Regularised Movie + User + Genres Effect - FINAL	0.8644514

As can be seen, the inclusion of an increasing number of relevant parameters, accounting for the movie, user and genre bias achieves a reduction in model error and RMSE. Furthermore, regularisation, which penalises large estimates derived from small samples, shows a positive effect on further improving model accuracy.

Conclusion

The objective of this report is to identify a suitable algorithmn to predict movie rataings, in order to be able to build an effective recommendation system. The model is trained using a separate dataset, which itself is partitioned into both training and testing datasets. This allowed for the selection of appropriate parameters

of interest, including λ as a tuning parameter through cross-validation. The performance of each model was assessed according to the RMSE, reflecting the typical error made when predicting a movie rating.

The final model, selected as that with the lowest RMSE captured the bias contributed by each movie, user, as well as genre category. Once selected, this was then tested, considered as a final evaluation of performance, against a separate validation dataset. This yielded an RMSE of 0.864.

In handling such a large dataset, the chosen linear model method of analysis was somewhat limited for this report, where more complex models could potentially glean greater insights and go further in reducing RMSE.

A recommendation system such as this, employing machine learning to predict ratings, has a wide degree of application and allows for extension to other areas of work. Further models could be developed to predict for example product ratings, for various different e-commerce platforms. Given data regarding users, products and product categories, ratings could be predicted to deliver similar effective customer recommendation systems, to optimise customer experience and engagement as well as company sales.