Movies project

James Edholm

15 October 2020

Introduction

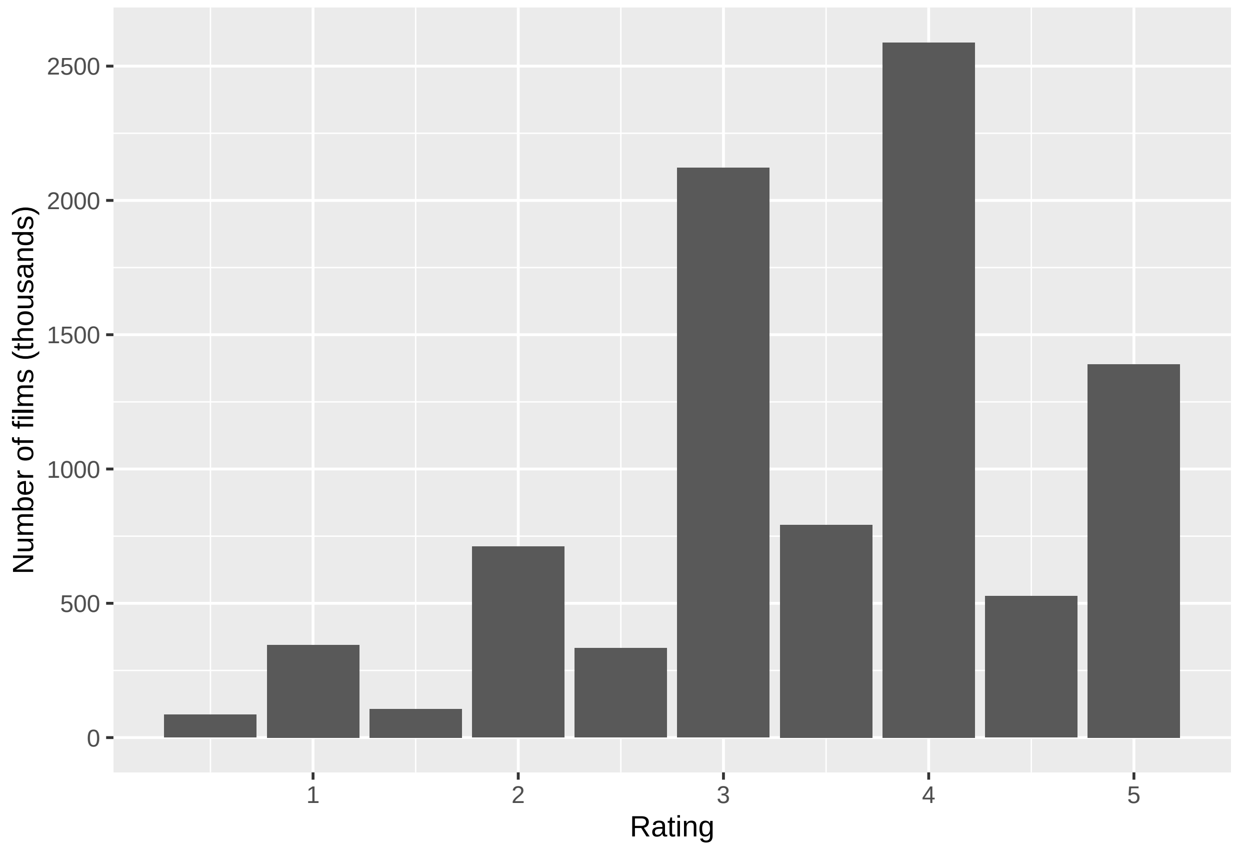
I use a data set of 9 million movie ratings and the user, the movie title and the time of the rating to construct a model which predicts the ratings of other movies given similar information. I investigate which methods produce the best results by splitting the data set into training and test sets, and optimising the model based on how well it predicts the ratings of the test set.

Overview of data set

The edx movies data set contains 9,000,055 ratings of 10,677 different movies. The ratings cover films with genres including Action, Adventure, Animation, Children, Comedy, Crime, Drama, Fantasy, Musical, Sci-Fi, Thriller and War. Each user is given a unique user ID and each film is given a unique movie ID. The timestamp of the review and the title of the film are also provided.

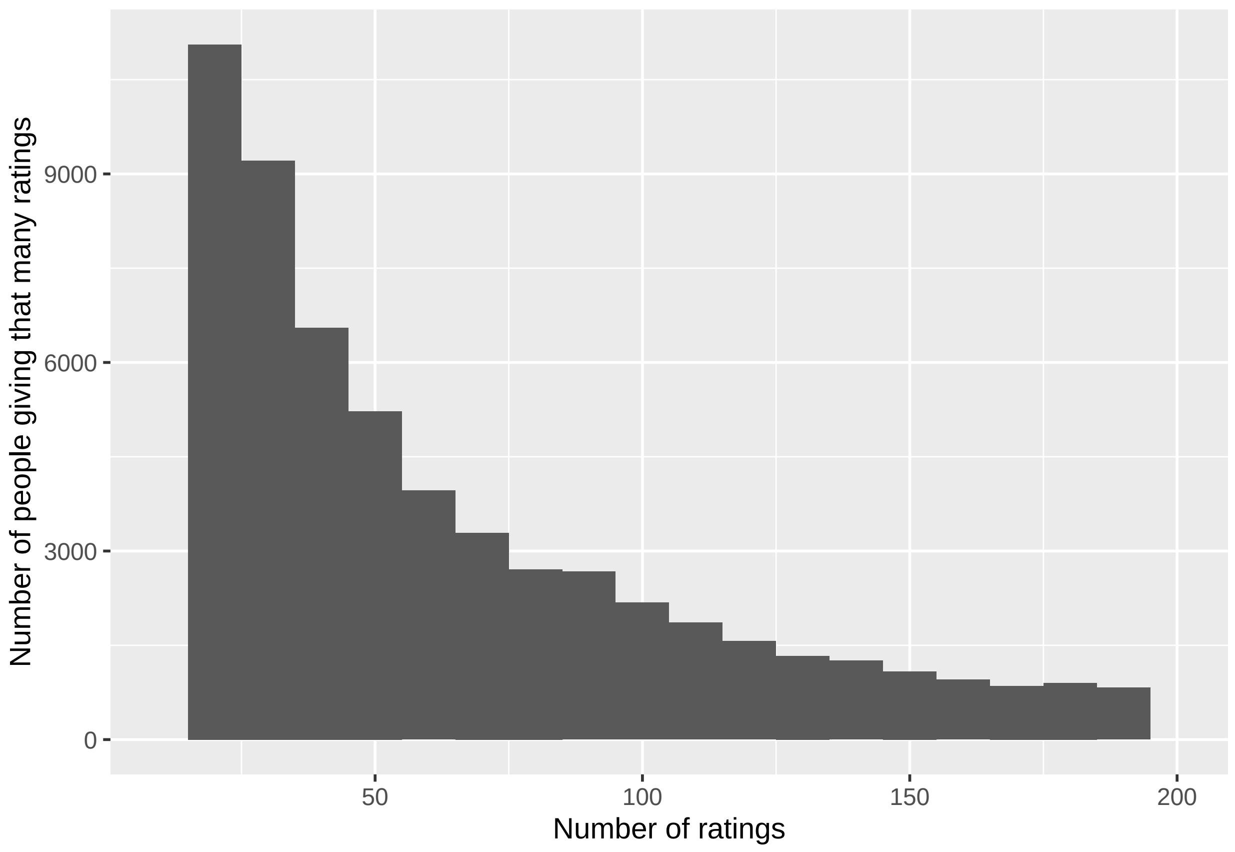
We can plot the distribution of the ratings, showing that 3 and 4 are the most popular ratings, and people tend to give integer ratings rather than half stars, e.g. 2.5 or 3.5.

```r
#Group films by rating, and calculate number of films with that rating (in thousands)
edx %>%
group_by(rating) %>%
summarise(number=n()/1000) %>%
ggplot(aes(x=rating, y=number))+
scale_y_continuous(breaks = c(seq(0, 28000, 500))) +
geom_bar(stat="identity") +
labs(x="Rating", y="Number of films (thousands)")
```



We can also plot the distribution of users by how many ratings they gave, showing that most users in the data set had rated fewer than a hundred movies.

```r
#Group ratings by user, and show distribution of users by number of ratings
edx %>%
  group_by(userId) %>%
  summarise(number_reviews=n()) %>%
  ggplot(aes(number_reviews))+
  geom_histogram(binwidth= 10)+
  scale_x_continuous(limits=c(10,200)) +
  labs(x="Number of ratings", y="Number of people giving that many ratings")
```



Methods

Introduction to method

I will use the machine learning packages to investigate whether we can predict the rating given, based on the user ID, movie ID, timestamp and genres. Before doing any training, I would expect that certain categories would be better predictors of rating than others.

For example, certain users probably give significantly higher ratings on average to films than other users, and certain films are probably likely to gain much higher ratings than other films. I think the genre would probably have a small effect on the rating given. I anticipate that both the year of the rating and the time of day it was given might have a (very small) effect on the rating - people might have become more harsh over the years, and they might also be more likely to give low ratings when they are sleepy!

Following the suggestion from the Netflix prize winners, I also think that the time since a user's first review will be important, as well as the time since a film's review, since users might well become harsher critics since their first review and those who review a film immediately are likely to be those who wanted to go see it anyway.

Methods

I will try to investigate the Root Mean Square Error (RMSE) given by different methods. We want to minimise the RMSE, which represents the difference between the predicted rating and the actual rating.

Using the provided code, I created the edx data set and the validation data set. The validation data set contains 999,999 ratings (i.e. 10% of the edx data set).

I started by splitting the edx data into training data (edx_train) and test data (edx_test). I then tried several models in order to find the best method. Unless otherwise stated, the RMSE is for predictions for the test data *train_set* based on the training data *test_set*.

```r
test_i <- createDataPartition(y = edx$rating, times = 1, p = 0.3, list = FALSE)
train_set <- edx[-test_i,]
test_set <- edx[test_i,]
test_set <- test_set %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

nrow(train_set) #print number of entries of the training set
```

```
## [1] 6300037
```

```r
nrow(test_set) #print number of entries of the test set
```

```
## [1] 2699928
```

```r
mu <- mean(train_set$rating) #find mean of training set ratings
```

Only including the overall average

I started by simply predicting that each movie had a rating which was the average of the training data μ. This gave an error compared to the actual rating of 1.061. This is the RMSE, the average error compared to the true value.

```r
#Predicted purely based on average rating of entire training set
predicted_ratings <- test_set %>%
  mutate(pred = mu ) %>%
  pull(pred)
rmse_mu <- RMSE(predicted_ratings, test_set$rating)
rmse_mu
```

```
## [1] 1.060597
```

Corrections for films

I introduced a correction by including simply the movie rating, i.e. if the average rating given to a certain movie was 3.0, then the prediction for ratings of that movie would be 3.0. This generated a relatively high RMSE of 0.9444.

```r
#Just include average rating of each film
movie_avgs <- train_set %>%
  group_by(movieId) %>% #group by movie
  summarize(b_i = mean(rating - mu)) #add a term b_i which adjusts rating based on movie

#produce predictions based on adjusting depending on movie
predicted_ratings <- test_set %>%
  left_join(movie_avgs, by='movieId') %>%
  mutate(pred = mu + b_i) %>%
  pull(pred)
```

```
rmse_movs <- RMSE(predicted_ratings, test_set$rating) #test how good this prediction is
rmse_movs
```

```
## [1] 0.9443758
```

Corrections for users

Next, I also included a term to account for certain users giving higher or lower ratings. If a user had given lower ratings to certain movies than the average ratings of those movies, then they would probably also rate other movies lower than expected. The same would apply for users giving higher ratings than expected. I therefore added a correction to the previous method, so that for each user I calculated the average difference from the expected rating, $b_u = rating - \mu - b_i$, and then added that to the prediction. Using this more sophisticated method gave an RMSE of 0.8673, a significant improvement.

```
#Include a term b_u which adjusts rating based on user giving rating
user_avgs <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  group_by(userId) %>%       #group by user
  summarize(b_u = mean(rating - mu - b_i)) #add a term b_u which adjusts
                                           #rating based on user

#Predict ratings based on film and user
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)

rmse_user <- RMSE(predicted_ratings, test_set$rating) #test how good this prediction is
rmse_user
```

```
## [1] 0.8673029
```

Corrections for genres

I next investigated whether adding genres to the prediction would improve our estimate. In the introduction, I guessed that adding genres would not significantly improve the estimate, because we are already correcting for the specific film. When I included genres, the RMSE reduced very slightly to 0.8670. Note that I only corrected for the specific combinations of genres, rather than correcting for each genre separately, which could be future work.

```
#Include genres
genre_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)) #add term to adjust for
                                                 #popularity of that combination of genres

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

```r
  pull(pred)

rmse_genres <- RMSE(predicted_ratings, test_set$rating) #test how good this prediction is
rmse_genres
```

```
## [1] 0.8669695
```

###Corrections for year/time of day I tested whether including the timestamp could help improve the rating. The timestamp is given in the format of seconds since 00.00 on 1st January 1970. As there are unlikely to be multiple ratings in the same second, I instead decided to test both whether the year of the rating and the hour of the day had any effect on the rating.

However, including the year or the time of day did not change the RMSE significantly, which was 0.8670 in both cases. Interestingly, there were two ratings in my test set which were from 1995, but the earliest rating from the training set was from 1995. This caused an error because the method had not produced an adjustment term for ratings from 1995, so the prediction was NA. I solved this by removing the adjustment term if it was NA.

```r
#Include year
year_set <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(rating_year= floor(timestamp/(3600*24*365))+1970) %>% #define the rating year
  group_by(rating_year) %>%
  summarize(b_y = mean(rating - mu - b_i - b_u-b_g)) #add a term b_y which adjusts
                                                     #rating based on year

#predict ratings of test set
predicted_ratings <- test_set %>%
  mutate(rating_year= floor(timestamp/(3600*24*365))+1970) %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(year_set, by='rating_year') %>%
  mutate(pred=ifelse(           #predict values, ignoring b_y if it is NA
    is.na(b_y),
    mu + b_i + b_u + b_g,
    mu + b_i + b_u + b_g + b_y)) %>%
  pull(pred)

RMSE(predicted_ratings, test_set$rating) #test how good this prediction is
```

```
## [1] 0.866953
```

```r
#Instead include time of day
hour_set <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(rating_hour= round((timestamp/3600) %% 24)) %>% #define hour of day
  group_by(rating_hour) %>%
  summarize(b_h = mean(rating - mu - b_i - b_u-b_g, na.rm=TRUE))
  #add a term b_h which adjusts rating based on hour

#predict ratings of test set
```

```r
predicted_ratings <- test_set %>%
  mutate(rating_hour= round((timestamp/3600) %% 24)) %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  left_join(hour_set, by='rating_hour') %>%
  mutate(pred = mu + b_i + b_u + b_g + b_h) %>%
  pull(pred)

RMSE(predicted_ratings,test_set$rating) #test how good this prediction is
```

```
## [1] 0.8669645
```

Adjusting for months since film's first review

Following the suggestion in the Netflix prize papers, I added a term to correct for the number of months since the film's first review. This enabled the model to take into account that people who review a film shortly after it is released are likely to be people who were keen to see the film and therefore may rate it higher. Adding this effect reduced the RMSE to 0.8669, although when I later included regularisation it further shrank the estimate.

```r
#Number of months since first rating of film
months_film_set <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(movieId) %>%
  mutate(months_since_film=floor((timestamp - min(timestamp))/(30*24*3600))) %>%
  ungroup() %>%
  group_by(months_since_film) %>%
  summarize(b_f = mean(rating - mu - b_i -b_u, na.rm=TRUE))
      #add a term b_f which adjusts based on time since film's first rating

#predict ratings of test set
predicted_ratings <- test_set %>%
  group_by(movieId) %>%
  mutate(months_since_film=floor((timestamp - min(timestamp))/(30*24*3600))) %>%
  ungroup() %>%
  group_by(userId) %>%
  mutate(sqrt_months_since_rating=
      floor(sqrt(abs((timestamp - min(timestamp))/(30*24*3600))))) %>%
  ungroup() %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(months_film_set, by='months_since_film') %>%
  mutate(pred = mu + b_i + b_u + b_f) %>%
  pull(pred)
rmse_movie_time <- RMSE(predicted_ratings,test_set$rating, na.rm=TRUE)
rmse_movie_time #test how good this prediction is
```

```
## [1] 0.8669378
```

I thought that choosing months as a time (as opposed to days or years for example) was a reasonable compromise between having bins that were too small and too big.

Adjusting for months since user's first review

Following the suggestion in the Netflix prize papers, I added a term to correct for the number of months since the users first review, since users may get harsher since the first film they had reviewed. Adding this effect did not have a significant effect on the RMSE, which remained at 0.8670, although when I later included regularisation it again shrank the estimate.

```r
#Number of months since user's first rating
months_rating_set <- train_set %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(movieId) %>%
  mutate(months_since_film=floor((timestamp - min(timestamp))/(30*24*3600))) %>%
  ungroup() %>%
  left_join(months_film_set, by='months_since_film') %>%
  group_by(userId) %>%
  mutate(sqrt_months_since_rating=
      floor(sqrt(abs((timestamp - min(timestamp))/(30*24*3600))))) %>%
  ungroup() %>%
  group_by(sqrt_months_since_rating) %>%
  summarize(b_d = mean(rating - mu - b_i - b_u, na.rm=TRUE))
  #add a term b_d which adjusts based on time since user's first rating

#predict ratings of test set
predicted_ratings <- test_set %>%
  group_by(movieId) %>%
  mutate(months_since_film=floor((timestamp - min(timestamp))/(30*24*3600))) %>%
  ungroup() %>%
  group_by(userId) %>%
  mutate(sqrt_months_since_rating=
      floor(sqrt(abs((timestamp - min(timestamp))/(30*24*3600))))) %>%
  ungroup() %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(months_film_set, by='months_since_film') %>%
  left_join(months_rating_set, by='sqrt_months_since_rating') %>%
  mutate(pred = mu + b_i + b_u + b_d + b_f) %>%
  pull(pred)

rmse_user_time <-RMSE(predicted_ratings,test_set$rating, na.rm=TRUE)
#test how good this prediction is
rmse_user_time
```

```
## [1] 0.8669118
```

Regularisation

We trained our method on movies with a wide variety of ratings, including some that only had one rating. Estimates for these films should not be trusted, because the low numbers of ratings increases the uncertainty.

Rather than just taking these estimates, it is normally better to make a prediction that is closer to the mean of all the movies. Regularisation is the process of penalising correction terms that are generated by a low number of entries (low number of ratings in this case).

The penalty is given by adding an extra term to the sum of squares, which punishes ratings which are both far away from the mean and based on a small amount of data. In order to minimise this new equation, we use a new b_i, given by

$\hat{b}_i = \frac{\sum(\text{rating}-\mu)}{n_i+\lambda}$

where λ is a tuning parameter and n_i is the number of ratings in the category i.

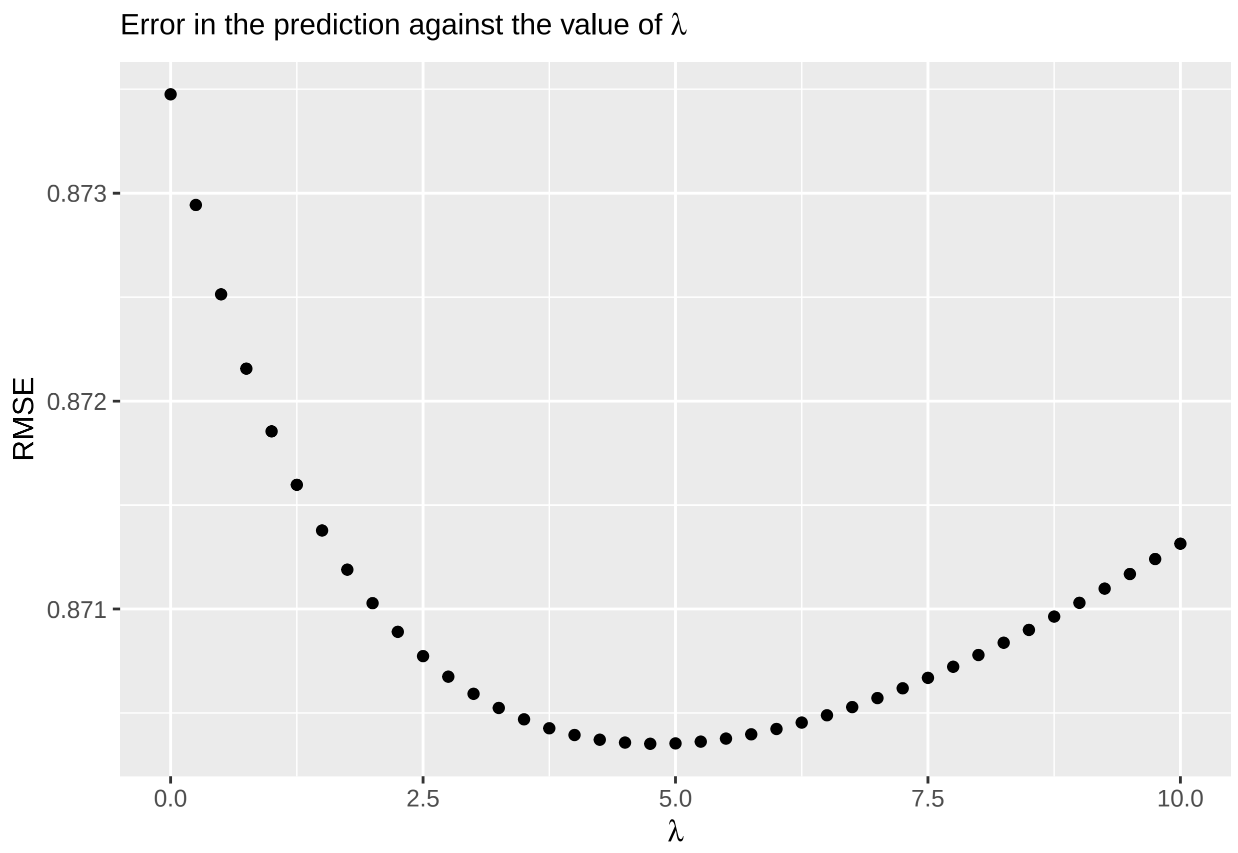
Regularisation method

I split the training data up into two sets, with training_set_1 representing 70% of the total, then tested which value of λ minimised the RMSE (by training on the first part of the training set with several values of λ and then testing on the second part). I started by taking values of λ between 0 and 10, with gaps of 0.25. This showed that the RMSE was minimised when $\lambda = 4.75$.

```
#split training data up to tune lambda
test_i <- createDataPartition(y = train_set$rating, times = 1, p = 0.5, list = FALSE)
train_set_1 <- train_set[-test_i,]
train_set_2 <- train_set[test_i,]
train_set_2 <- train_set_2 %>%
  semi_join(train_set_1, by = "movieId") %>%
  semi_join(train_set_1, by = "userId")

#Set a wide range for lambdas
lambdas <- seq(0, 10, 0.25)
#Find out the RMSE for each value of lambda
rmses <- sapply(lambdas, function(l){

  mu <- mean(train_set_1$rating)

  #Term to correct for more or less popular movies
  b_i <- train_set_1 %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - mu)/(n()+l))
  #add b_i which adjusts for popularity of movie

  #Term to correct for more or less harsh users
  b_u <- train_set_1 %>%
    left_join(b_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - b_i - mu)/(n()+l))
  #add b_u which adjusts for harshness of users

  #Adjust rating depending on number of months since film's first rating
  months_film_set <- train_set_1 %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    group_by(movieId) %>%
    mutate(months_since_film=floor((timestamp - min(timestamp))/(30*24*3600))) %>%
    ungroup() %>%
    group_by(months_since_film) %>%
    summarize(b_f = sum(rating - mu - b_i -b_u)/(n()+l))
  #add b_f which adjusts for time since film's first rating

  #Adjust rating depending on number of months since user's first rating
  months_rating_set <- train_set_1 %>%
    group_by(movieId) %>%
    mutate(months_since_film=floor((timestamp - min(timestamp))/(30*24*3600))) %>%
    ungroup() %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(months_film_set, by='months_since_film') %>%
    group_by(userId) %>%
    mutate(sqrt_months_since_rating
           =floor(sqrt(abs((timestamp - min(timestamp))/(30*24*3600))))) %>%
    ungroup() %>%
    group_by(sqrt_months_since_rating) %>%
    summarize(b_d = sum(rating - mu - b_i - b_u - b_f)/(n()+l))
  #add b_d which adjusts for time since user's first rating

  #Predict ratings for train_set_2
  predicted_ratings <- train_set_2 %>%
    group_by(movieId) %>%
    mutate(months_since_film=floor((timestamp - min(timestamp))/(30*24*3600))) %>%
    ungroup() %>%
    group_by(userId) %>%
    mutate(sqrt_months_since_rating
           =floor(sqrt(abs((timestamp - min(timestamp))/(30*24*3600))))) %>%
    ungroup() %>%
    left_join(b_i, by='movieId') %>%
    left_join(b_u, by='userId') %>%
    left_join(months_rating_set, by='sqrt_months_since_rating') %>%
    mutate(pred = mu + b_i + b_u + b_d ) %>%
    pull(pred)

  RMSE(predicted_ratings, train_set_2$rating, na.rm=TRUE) #test how good prediction is
})

lambda <- lambdas[which.min(rmses)] #find the value of lambda which minimises the RMSE
lambda
```

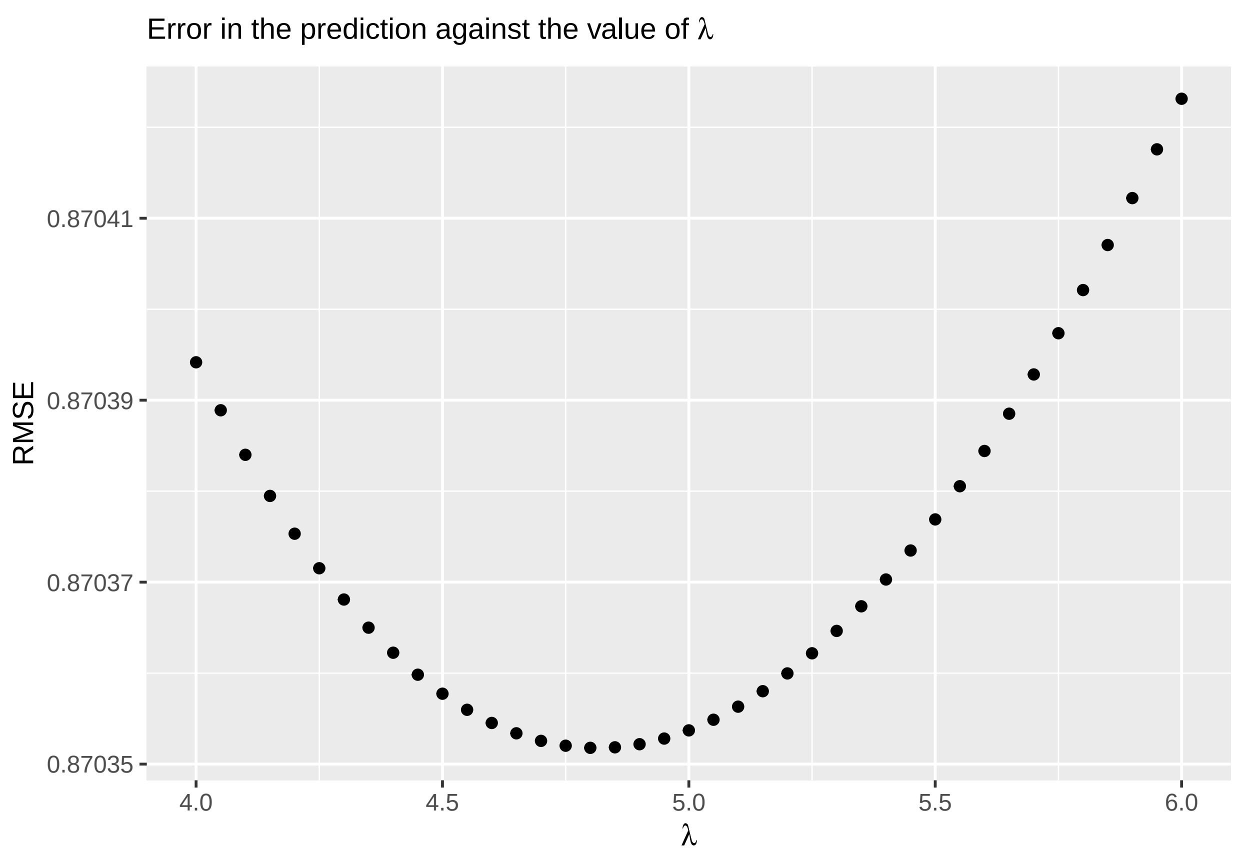
```
## [1] 4.75
```

```
rmse_wide_range  <- data.frame(lambdas, rmses)

ggplot(data=rmse_wide_range, aes(x=lambdas, y=rmses))+ #plot the error in the prediction
  geom_point() +                                       #against the value of lambda
  labs(x=expression(paste(lambda)), y= "RMSE",
       subtitle= expression(paste("Error in the prediction against the value of ", lambda)))
```



I then took a narrower range of λ between 4 and 6, with smaller gaps of 0.05, to find the exact value of λ best suited for our model. This showed that the model performed best when $\lambda = 4.8$.

```
rmse_narrow_range  <- data.frame(lambdas, rmses)
ggplot(data=rmse_narrow_range, aes(x=lambdas, y=rmses))+
  geom_point() +
  labs(x=expression(paste(lambda)), y= "RMSE",
       subtitle= expression(paste("Error in the prediction against the value of ", lambda)))
```



I then used $\lambda = 4.8$ on the test set, which gave an RMSE of 0.8663.

```
#take mean of training set
mu <- mean(train_set$rating)

#correct for popularity of movie
b_i <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = sum(rating - mu)/(n()+lambda))

#correct for harshness of user
b_u <- train_set %>%
  left_join(b_i, by="movieId") %>%
  group_by(userId) %>%
  summarize(b_u = sum(rating - b_i - mu)/(n()+lambda))

#Adjust for number of months since film's first rating
months_film_set <- train_set %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  group_by(movieId) %>%
  mutate(months_since_film=floor((timestamp - min(timestamp))/(30*24*3600))) %>%
  ungroup() %>%
  group_by(months_since_film) %>%
  summarize(b_f = sum(rating - mu - b_i -b_u)/(n()+lambda))

#Number of months since user's first rating
```

```r
months_rating_set <- train_set %>%
  group_by(movieId) %>%
  mutate(months_since_film=floor((timestamp - min(timestamp))/(30*24*3600))) %>%
  ungroup() %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(months_film_set, by='months_since_film') %>%
  group_by(userId) %>%
  mutate(sqrt_months_since_rating=
           floor(sqrt(abs((timestamp - min(timestamp))/(30*24*3600))))) %>%
  ungroup() %>%
  group_by(sqrt_months_since_rating) %>%
  summarize(b_d = sum(rating - mu - b_i - b_u - b_f)/(n()+lambda))

#Predict ratings
predicted_ratings <- test_set %>%
  group_by(movieId) %>%
  mutate(months_since_film=floor((timestamp - min(timestamp))/(30*24*3600))) %>%
  ungroup() %>%
  group_by(userId) %>%
  mutate(sqrt_months_since_rating=
           floor(sqrt(abs((timestamp - min(timestamp))/(30*24*3600))))) %>%
  ungroup() %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(months_rating_set, by='sqrt_months_since_rating') %>%
  mutate(pred = mu + b_i + b_u + b_d ) %>%
  mutate(pred=ifelse(pred>5,5,pred)) %>% #Making sure all ratings stay within range
  mutate(pred=ifelse(pred<0,0,pred)) %>%
  pull(pred)

rmse_reg <- RMSE(predicted_ratings, test_set$rating, na.rm=TRUE)
#test how good this prediction is
rmse_reg
```

```
## [1] 0.8663501
```

Considering the range of ratings

In reality, the maximum rating is five and the minimum is zero, but our predicted ratings sometimes go slightly outside of this range (e.g. -0.2 or 5.1). By using the code

```r
mutate(pred=ifelse(pred>5,5,pred)) %>%
mutate(pred=ifelse(pred<0,0,pred)) %>%
```

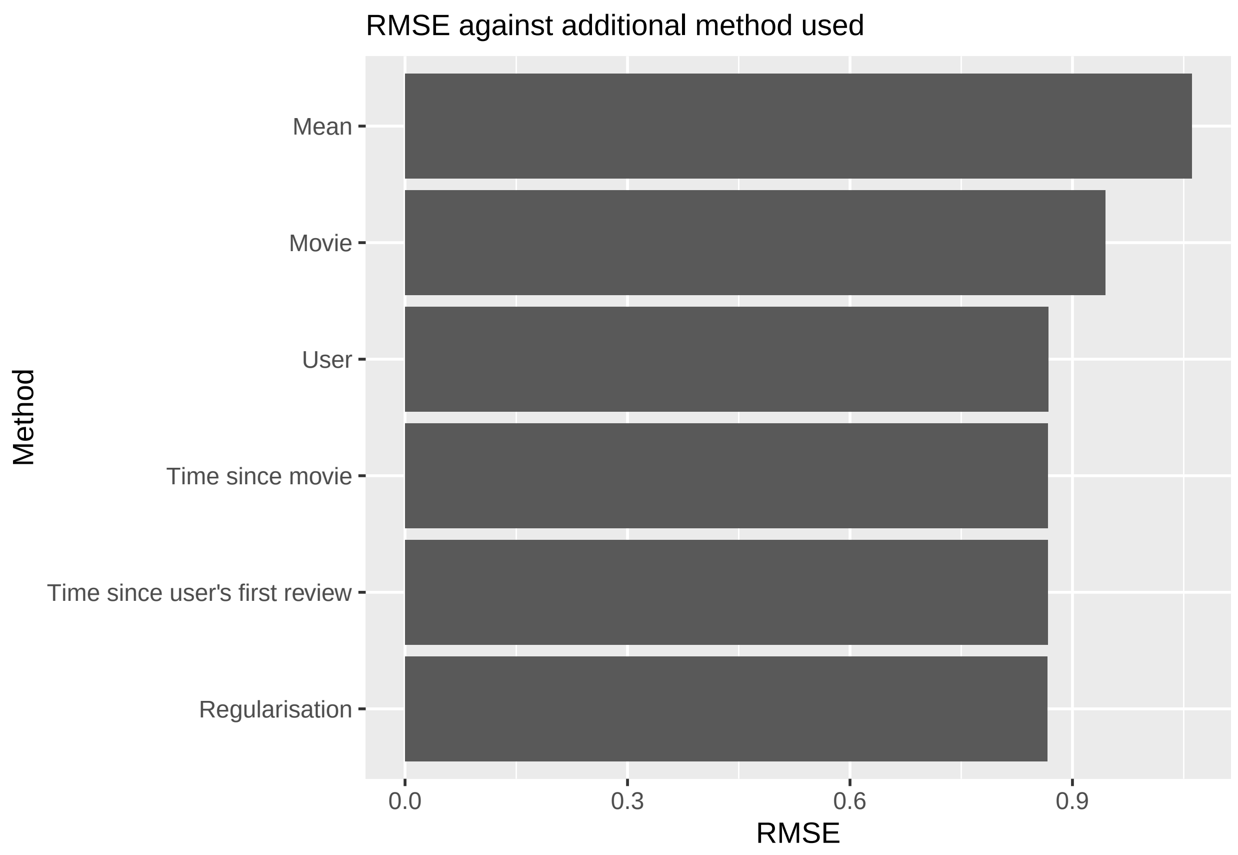
I ensured that all predictions were between zero and five. This did not significantly affect the RMSE, because the vast majority of predictions were already in this range, and those that were outside were only slightly outside.

Results

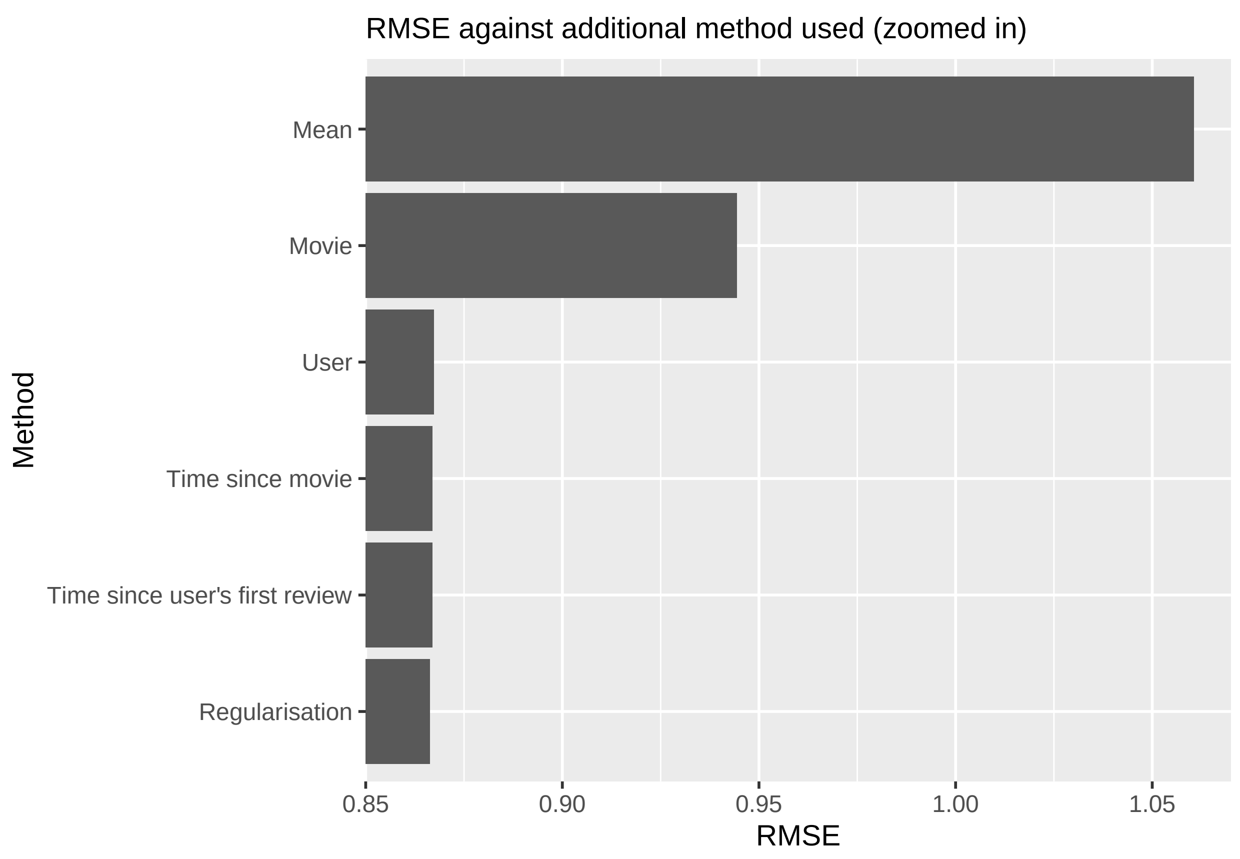
To predict the rating of a certain movie, I started with the mean of the training data. For my final model, I then corrected for the film, the particular user, the number of months since the film's first review and finally the number of months since the user's first rating. The first two corrections produced the most improvement, and the last two produced only a small improvement. The final improvement to my model was to include regularisation, which produced a moderate improvement.

I show how each additional method improved the RMSE in the following bar charts.

```r
names_rmse <- c("Mean", "Movie", "User", "Time since movie",
                "Time since user's first review", "Regularisation")
results_rmse <- c(rmse_mu, rmse_movs, rmse_user, rmse_movie_time,
                  rmse_user_time, rmse_reg)
df <- data.frame(names_rmse, results_rmse)
ggplot(data=df, aes(x= reorder(names_rmse,  results_rmse), y=results_rmse)) +
  geom_bar(stat="identity") +
  coord_flip() +
  labs(x="Method", y="RMSE", subtitle="RMSE against additional method used")
```



```r
ggplot(data=df, aes(x= reorder(names_rmse,  results_rmse), y=results_rmse)) +
  geom_bar(stat="identity") +
  coord_flip(ylim=c(0.86,1.06)) +
  labs(x="Method", y="RMSE", subtitle="RMSE against additional method used (zoomed in)")
```



I validated my method on the validation data. I again used regularisation with $\lambda = 4.8$, which gave an RMSE of 0.8642.

```
#Regularisation with lambda=4.8
mu <- mean(edx$rating)
l <- 4.8
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))


#Number of months since first rating of film
months_film_set <- edx %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  group_by(movieId) %>%
  mutate(months_since_film=floor((timestamp - min(timestamp))/(30*24*3600))) %>%
  ungroup() %>%
  group_by(months_since_film) %>%
  summarize(b_f = sum(rating - mu - b_i -b_u)/(n()+l))

#Number of months since user's first rating
```

```r
months_rating_set <- edx %>%
  group_by(movieId) %>%
  mutate(months_since_film=floor((timestamp - min(timestamp))/(30*24*3600))) %>%
  ungroup() %>%
  left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(months_film_set, by='months_since_film') %>%
  group_by(userId) %>%
  mutate(sqrt_months_since_rating
         =floor(sqrt(abs((timestamp - min(timestamp))/(30*24*3600))))) %>%
  ungroup() %>%
  group_by(sqrt_months_since_rating) %>%
  summarize(b_d = sum(rating - mu - b_i - b_u - b_f)/(n()+l))

#Predict ratings
predicted_ratings <- validation %>%
  group_by(userId) %>%
  mutate(sqrt_months_since_rating
         =floor(sqrt(abs((timestamp - min(timestamp))/(30*24*3600))))) %>%
  ungroup() %>%
  group_by(movieId) %>%
  mutate(months_since_film=floor((timestamp - min(timestamp))/(30*24*3600))) %>%
  ungroup() %>%
    left_join(b_i, by='movieId') %>%
  left_join(b_u, by='userId') %>%
  left_join(months_rating_set, by='sqrt_months_since_rating') %>%
  left_join(months_film_set, by='months_since_film') %>%
  mutate(pred = mu + b_i + b_u + b_d + b_f) %>%
  mutate(pred=ifelse(pred>5,5,pred)) %>%
  mutate(pred=ifelse(pred<0,0,pred)) %>%
  pull(pred)

RMSE(predicted_ratings, validation$rating, na.rm=TRUE) #test how good this prediction is
```

```
## [1] 0.8642187
```

Conclusion

I produced a model which predicts what a user will rate a certain movie, based on the user's past reviews, other ratings of the film, and the time since both the film's first review and the user's first review. I then added a correction whereby predicted ratings which are produced only on a low number of ratings are moved closer to the mean (regularisation). This method produced an RMSE of 0.8642.

The limitations of my model are that it would only work for ratings where there are already existing ratings of the same film, by the same user, with the same time after the film's first review and the same time after the user's first review. Therefore it could not predict how a user would rate a new film, or how a new user would rate a film.

Future work could include varying the size of the bin used to adjust for time since the film's first review and the user's first review. It could also include adjusting for the specific interaction of the user and specific genres, e.g. if Pam likes horror films and dislikes romance films, then it could adjust for this.

Finally, the model could correct for each genre separately, rather than for the combinations of genres as I tried (which didn't produce any improvement to the model).