HarvardX Capstone Project MovieLens

Alec Banner

Introduction

In 2006, Netflix set a $1million challenge to the data science community to improve their recommender engine by 10%. Recommender engines are used to predict how many stars a given user will predict a specific movie based on how well that user has rated similar movies and how well the specific movie is rated compared to similar movies. In 2009 the winners were announced to be 'BelKor' an international, collaborative team who surpassed the 10% improvment mark using a gradient boosted decision tree to combine over 500 models.

The MovieLens 10M dataset

The original netflix dataset is unavailable to the public, however GroupLens have produced their own dataset containging 10 million enteries which has become the center of many data science projects. Since 2009, more advanced models have reduced the RMSE down to 0.7485 based on a Bayesian timeSVD++flipped model. In this project the goal is to generate a recommender model for the MovieLens 10M dataset, capable of achieving an RMSE bellow 0.86490.

Loading the dataset:

```
################################
# Create edx set, validation set
################################

# Note: this process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse
```

```
## -- Attaching packages --------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.0     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   0.8.5
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## Warning: package 'tibble' was built under R version 3.6.2
```

```
## Warning: package 'purrr' was built under R version 3.6.2
```

```
## -- Conflicts ------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 3.6.2
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: data.table
```

```
##
## Attaching package: 'data.table'
```

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
## The following object is masked from 'package:purrr':
##
##     transpose
```

```r
# 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 <- 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")

# Validation set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
# if using R 3.5 or earlier, use `set.seed(1)` instead
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)
```

```
## Joining, by = c("userId", "movieId", "rating", "timestamp", "title", "genres")
```

```
edx <- rbind(edx, removed)

rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

This creates two dataframes: edx, our trianing set and validation, our test set. The models will be built and any training solely carried out on the edx dataset with validation only being used to test the models at the end and produce the final RMSEs. The edx dataset contains 9000055 enteries for users who have rated a movie, with each row representing a single rating. There are six columns containing data on the rating including: userId, movieId, rating, timestamp, title and genres.

```
#display top 10 rows of data
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
```

We can see that for the edx dataset there are 69878 unique users and 10677 unique movies

```
#number of users and number of movies
edx %>%
  summarise(n_users = n_distinct(userId), n_movies = n_distinct(movieId))
```

```
##   n_users n_movies
## 1   69878    10677
```

If our dataset contained an entery for every user rating every movie then there would be 7.4608741×10^8 enteries. However our dataset contains just 9000055 enteries. Therefore, our dataset is sparse with most users not having rated most movies. It is therefore the goal of this project to fill in the blank spaces with predicted ratings.

Adding a year column:

```
#add publication year to edx and validation datasets
edx <- edx %>% mutate(year = as.numeric(str_sub(title, -5, -2)))
validation <- validation %>% mutate(year = as.numeric(str_sub(title, -5, -2)))
```

A column has been added to both the edx and validation datasets for the containing the publication year of the movie. This has been done by subsetting the last five characters throught the penultimate character of the title string which contains the publication date. We can see that within this dataset films from 1915 through 2008 are included.

```
#years which films are from
edx %>%
  summarise(earliest = year[which.min(year)], latest = year[which.max(year)])
```

```
##   earliest latest
## 1     1915   2008
```

Root Mean Squared Error (RMSE)

RMSE will be used to assess the quality of the models built. RMSE is a measure of the error in our predictions when compared to the validation set, with larger values of RMSE meaning a larger error in our predictions. Therefore, we are trying to minimise RMSE.

The formula for RMSE is as follows: $\mathbf{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(True_values - Predicted_values)^2}{N}}$

This is the square root of the mean of the difference between the predicted and actual values squared. In this project RMSE will be calculated using the following formula:

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

Methods

First Model

The simplest solution to this problem is to predict that all movies have the same rating and all the differences are due to random variation.

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

```r
#naive model
mu <- mean(edx$rating)
```

In this model the predicted value of each rating will be 3.5124652.

Movie effects

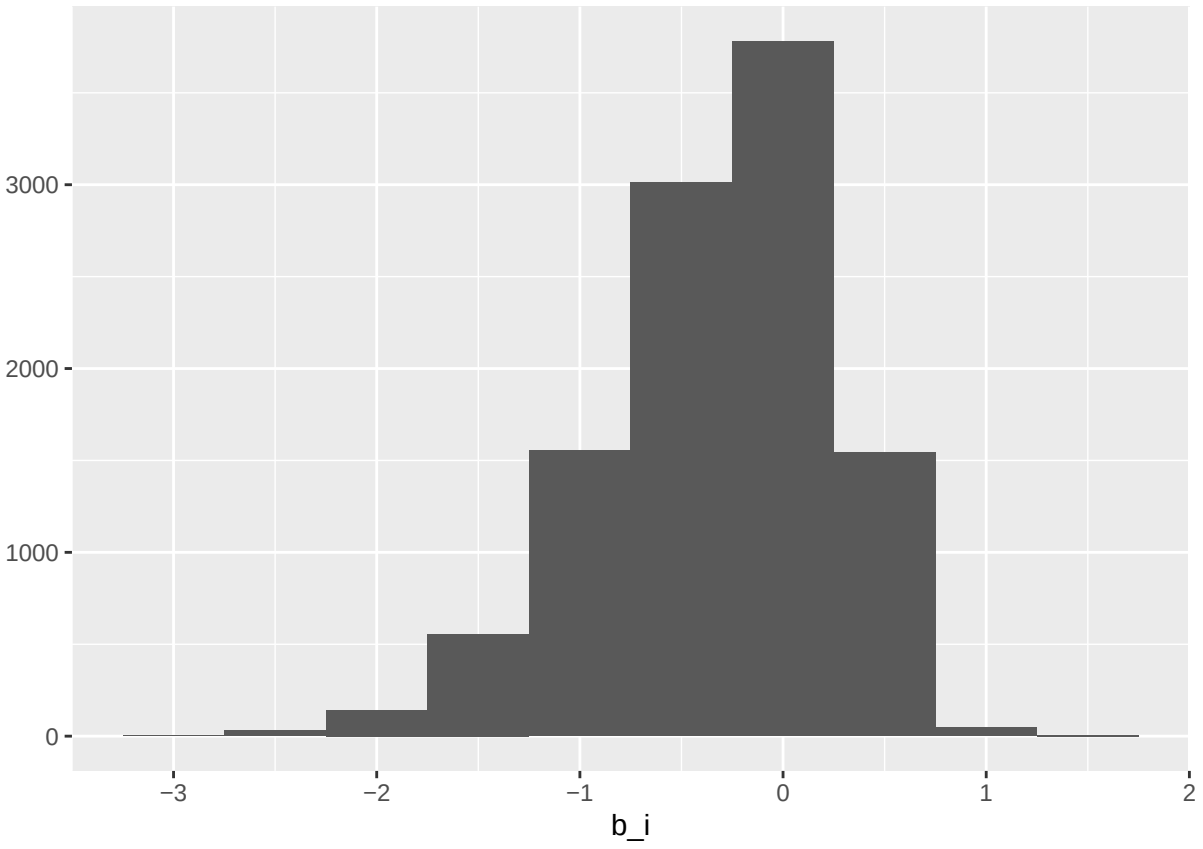
Next a model will be built which takes into account that, on average, some movies are rated higher than others, i.e. some movies are better than others, and therefore a new user is likely to also rate this movie higher than average.

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

```r
#movie model effects
movie_avgs <- edx %>%
  group_by(movieId) %>%
  summarise(b_i = mean(rating - mu))
```

If the dataset is grouped by the movieId, then b_i can be calculated as the mean of the difference between each rating and the average movie rating. If we plot b_i it becomes apparant that movie ratings vary and the resulting graph is screwed to the right and centered around the mean. The variation in b_i indicates that the movie effect should improve our recommender engine as each movie is not rated the same.

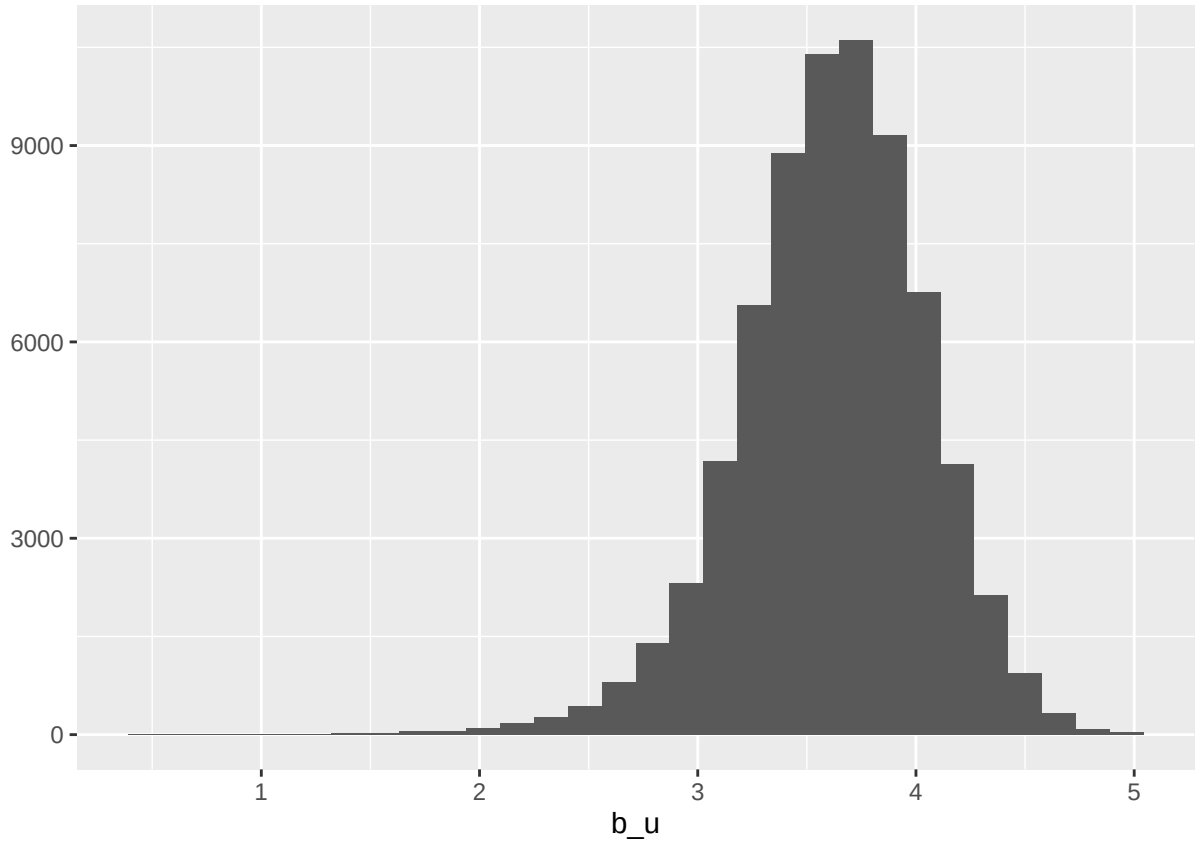
```r
#plot movie model effects
qplot(b_i, data = movie_avgs, bins = 10)
```



User effects

The third model will include the effects the user has on rating. In the first two models it was assumed that each user would rate each movie the same, however plotting b_u shows us that this is not the case.

```
#user effects
user_effects <- edx %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating))
#plot movie user effects
qplot(b_u, data = user_effects, bins = 30)
```



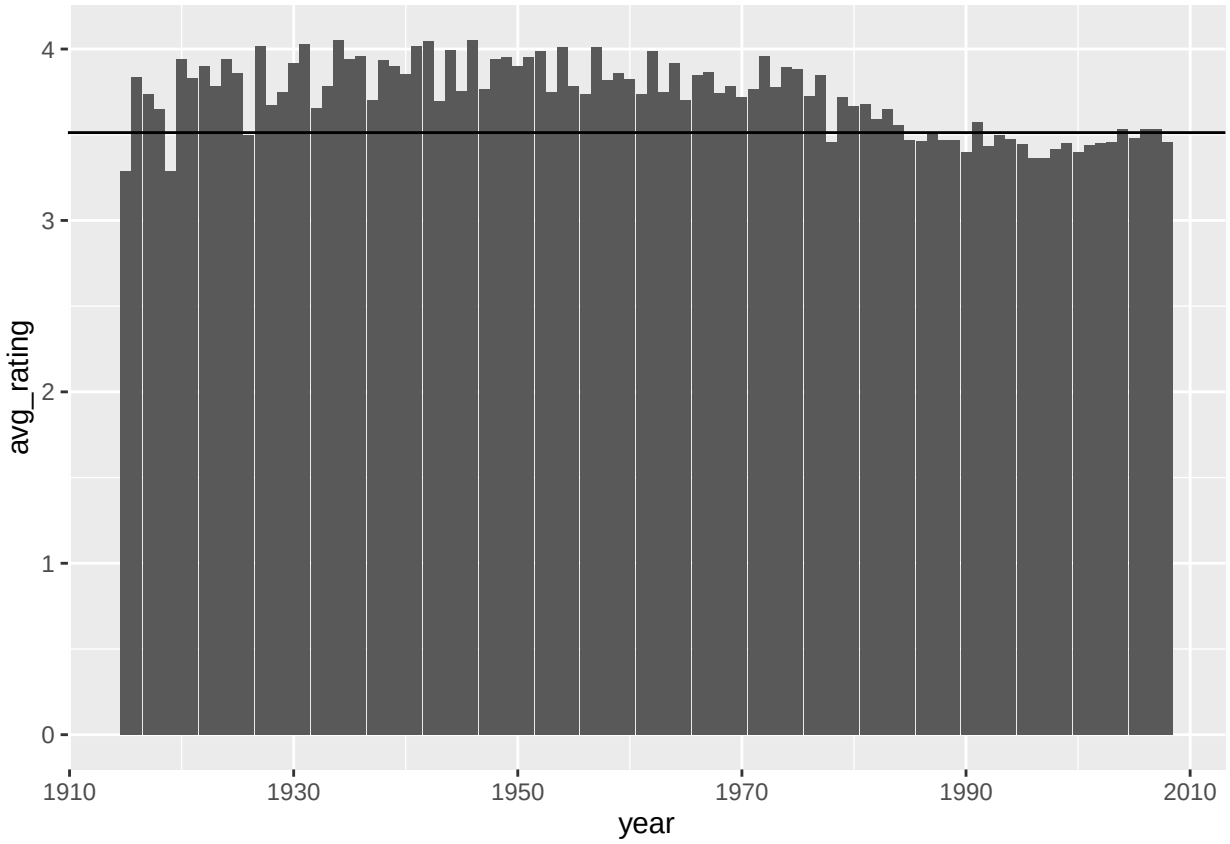
The plot shows that some users tend to rate movies much lower than average whilst other users tend to rate movies much higher. Again this is justification in including this factor in the final model.

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

Year effects

```
#year effects
year_effects <- edx %>%
  group_by(year) %>%
  summarise(avg_rating = mean(rating))
#plot year effects
year_effects %>%
```

```
  ggplot(aes(year, avg_rating)) +
  geom_bar(stat="identity") +
  geom_hline(yintercept = mu)
```



This plot shows that the average movie rating is not constant over time and moreover that movies since 1995 tend to be rated slightly bellow average whereas older movies tend to be slightly above average, indicating there maybe be a 'classic movie' factor in the ratings.

The year (b_y) effect will be included in the model as such:

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

Genre effect

Ideally a genre effect would have been evaluated for inclusion in this model as there is potential that some genres are more popular than others. This however was not possible as separating the genres of a movie containing multiple genres into mutliple rows in the data frame exceeded the memory available in the computer. Presented is the code which would separate the genres into multiple rows.

```
#genre effects

#separate movies under multiple genres into one row per genre
edx_genres <- edx %>% separate_rows(genres,sep = "\\|")
validation_genres <- validation %>% separate_rows(genres,sep = "\\|")
```

Regularisation

This dataset will contain some movies which have been rated more times than others and some users which have rated more movies than others. In cases where a movie has been rated very few times, the estimate is noisey and more prone to error due to a single particulary high or low estimate. Regularisation allows us to penalise estimates based on a small number of ratings.

Regularisation will be carried out using penalised least squares. To do this a random variable λ will be added to the model.

Looking at the movie effects as an example, the model can be reduced to:

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

Here when a movie has recieved a low number of ratings (n), λ will act to reduce its impact, whereas when a movie has recieved a high numer of ratings (n) the value of λ is neglegable.

In this case, regularisation will be implemented to affect all the effects not just movie rating.

As a random variable, the value of λ needs to be estimated. This will be done by spliting the edx dataset into training and testing sets, so as not to include the validation data at any point until the final predictions to prevent over fitting. λ will be estimated by calculating the RMSE for multiple values of λ on the test set and then plotting λ against RMSE to find the value which gives the smallest RMSE.

```
#split edx into training_set and test_set so that validation set isn't used for regularisation
set.seed(1, sample.kind = "Rounding")
```

```
## Warning in set.seed(1, sample.kind = "Rounding"): non-uniform 'Rounding' sampler
## used
```

```
test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, 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") %>%
  semi_join(train_set, by = "year")
```

Results

Assessing the RMSEs for each model compared to the validation set. The validation set is only being used for calculating RMSE and is not used for training in anyway

Naive model

```
#naive model predictions
naive_RMSE <- RMSE(validation$rating, mu)
```

The naive model, it is assumed every movie recieves the same rating. The RMSE of this model shows that this model is poor with an average error of greater than 1 star per movie

method	RMSE
naive	1.061202

Movie effects model

```
#make predictions based on movie model effects
predicted_ratings_movie <- mu + validation %>%
  left_join(movie_avgs, by = "movieId") %>%
  pull(b_i)

movie_model_RMSE <- RMSE(validation$rating, predicted_ratings_movie)
```

When incliuding the effects of the movie on this model, the RMSE is reduced.

method	RMSE
naive	1.0612018
Movie Model	0.9439087

User effects model

```
#user effects model
user_avgs <- edx %>%
  left_join(movie_avgs, by = "movieId") %>%
  group_by(userId) %>%
  summarise(b_u = mean(rating - mu - b_i))


predicted_ratings_movie_user <- validation %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  mutate(pred = mu + b_i + b_u) %>%
  pull(pred)

movie_user_model_RMSE <- RMSE(validation$rating, predicted_ratings_movie_user)
```

Inclusion of the user effect into the model again reduces the RMSE

method	RMSE
naive	1.0612018
Movie Model	0.9439087
User Model	0.8653488

Year effects

```
year_avgs <- edx %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  group_by(year) %>%
  summarise(b_y = mean(rating - mu - b_i - b_u))

predicted_ratings_movie_user_year <- validation %>%
  left_join(movie_avgs, by = "movieId") %>%
  left_join(user_avgs, by = "userId") %>%
  left_join(year_avgs, by = "year") %>%
  mutate(pred = mu + b_i + b_u + b_y) %>%
  pull(pred)

movie_user_year_model_RMSE <- RMSE(validation$rating, predicted_ratings_movie_user)
```

The effect of year on the overall model is much smaller than the effects of either movie or user. It doesnt have a negative impact on the model so will continue to be included in the model

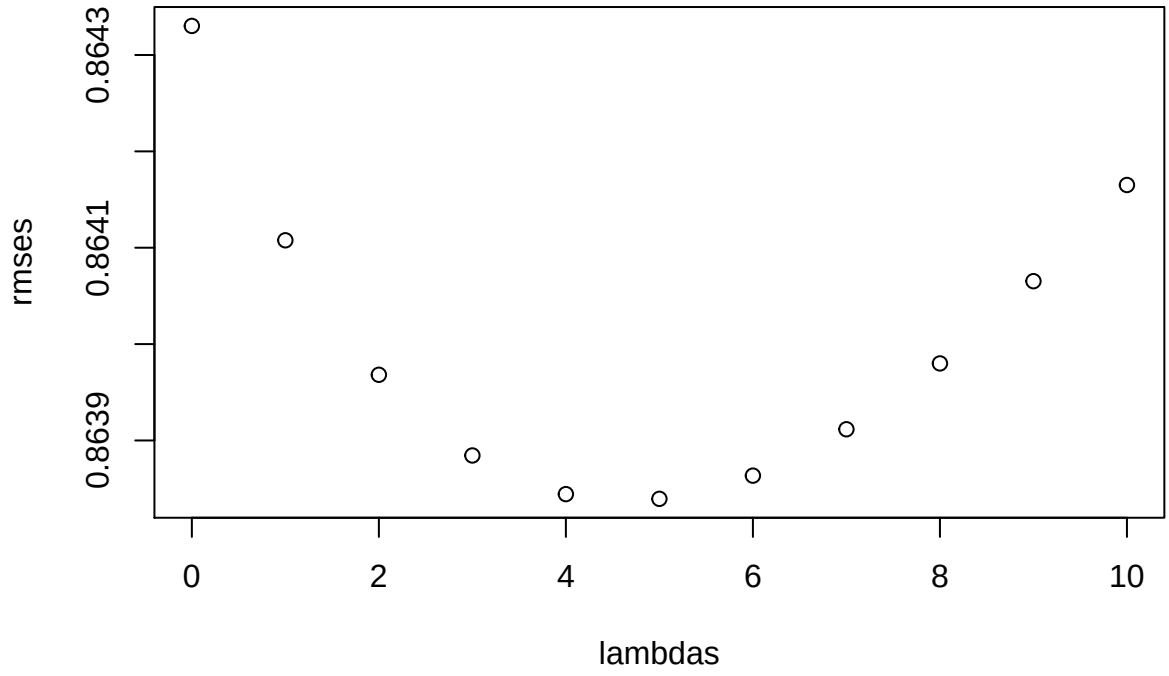
method	RMSE
naive	1.0612018
Movie Model	0.9439087
User Model	0.8653488
Year Model	0.8653488

Regularisation

Regularisation with be carried out on the final model using the training set, initially for integer λ between 1 and 10. The test set is used to assess the RMSE

```
lambdas <- seq(0, 10, 1)

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 - mu - b_i)/(n()+l))

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

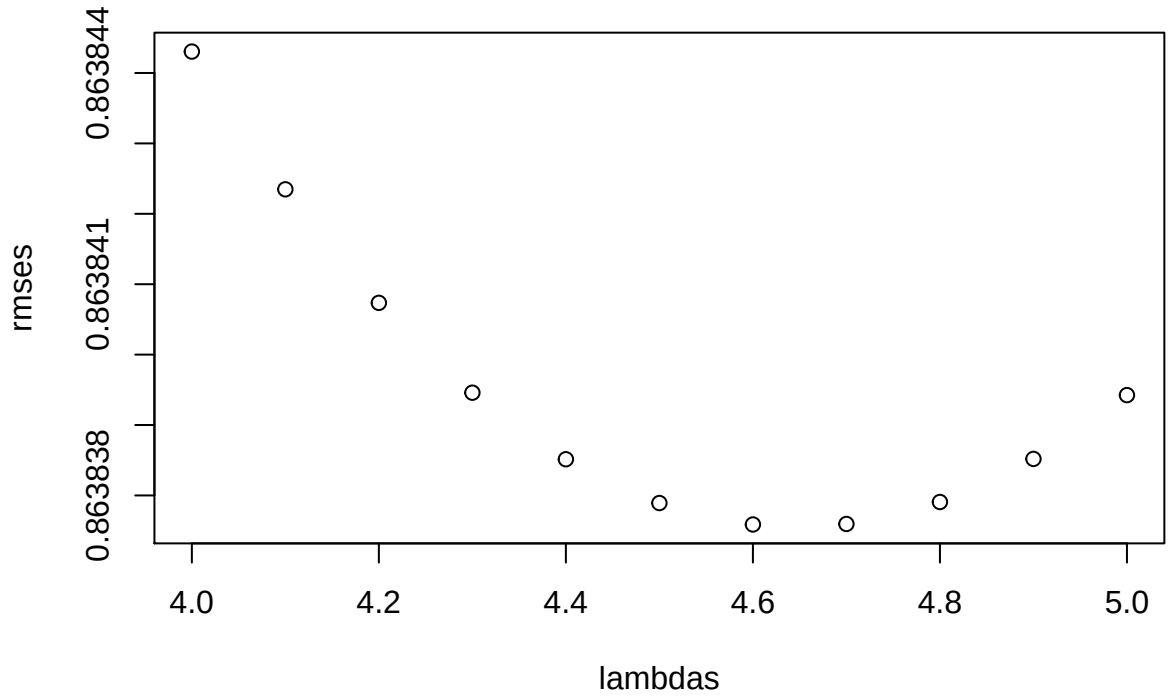
```
  predicted_ratings <- test_set %>%
    left_join(b_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    left_join(b_y, by = "year") %>%
    mutate(pred = mu + b_i + b_u + b_y) %>%
    pull(pred)

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

plot(lambdas, rmses)
```



looking at the plot of λ against rmse, the minimum RMSE is achieved with a value somewhere between 4 and 5. A second round of regularisation using λ in this range will be used to estimate a more accurate value of lambda.

```
#second round of regularisation
lambdas <- seq(4, 5, 0.1)
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 - mu - b_i)/(n()+l))

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

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

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

plot(lambdas, rmses)
```



From the second round of regularisation we can see that the minimum value of RMSE is achieved with a λ of 4.6. This is the value which will be used in the final model built using the edx dataset.

```r
#final Model


l <- lambdas[which.min(rmses)]
mu <- mean(edx$rating)

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

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

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

predicted_ratings_regularisation <- validation %>%
  left_join(b_i, by = "movieId") %>%
  left_join(b_u, by = "userId") %>%
  left_join(b_y, by = "year") %>%
  mutate(pred = mu + b_i + b_u + b_y) %>%
  pull(pred)


regularisation_model <- RMSE(validation$rating, predicted_ratings_regularisation)
```

We can see that using regualrisation in this final model has a greatly reduced RMSE compared to the previous models.

method	RMSE
naive	1.0612018
Movie Model	0.9439087
User Model	0.8653488
Year Model	0.8653488
Regulisation	0.8645233

Conclusions

The **RMSE** achieved by the final model is 0.8645233 this is bellow the target value of 0.86490 and therefore considered successful.

Future work

This model could however be improved further as is shown by the low RMSEs achieved by the best models on the MovieLens dataset. One approach would be to user User-Based Collaborative Filtering, a matrix

factorisation technique, which predicts that users with similar characteristics will have similar taste and rate movies similarly. These models can be built using the **recommenderlab** package.

In this project edx has been used as a dataframe, however recommenderlab requires data in a matrix type known as a "realRatingMatrix". Attempts were made to coerce data in the train_set and test_set, used for regularisation, into this format. However, the problem of data sparsity, the fact that the matrix produced is vary large but much of the data is empty, means that large amounts of memory and processing power are required. These requirements proved to be significantly greater than the hardware available in this project. Instead, a demonstration was produced using a much smaller subset of the data, only containing movies with more than 5000 ratings and users who had rated more than 100 movies. A test_matrix was created containing only movies in the test_set only contained within the recommender_matrix.

```
library(recommenderlab)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loading required package: arules

## Warning: package 'arules' was built under R version 3.6.2

##
## Attaching package: 'arules'

## The following object is masked from 'package:dplyr':
##
##     recode

## The following objects are masked from 'package:base':
##
##     abbreviate, write

## Loading required package: proxy

## Warning: package 'proxy' was built under R version 3.6.2

##
## Attaching package: 'proxy'

## The following object is masked from 'package:Matrix':
##
##     as.matrix

## The following objects are masked from 'package:stats':
##
##     as.dist, dist
```

```
## The following object is masked from 'package:base':
##
##     as.matrix

## Loading required package: registry

## Registered S3 methods overwritten by 'registry':
##   method               from
##   print.registry_field proxy
##   print.registry_entry proxy

##
## Attaching package: 'recommenderlab'

## The following object is masked _by_ '.GlobalEnv':
##
##     RMSE

## The following objects are masked from 'package:caret':
##
##     MAE, RMSE
```

```r
#create minimised train and test sets
train_recommender <- train_set %>%
  group_by(movieId) %>%
  filter(n()>5000) %>%
  ungroup() %>%
  group_by(userId) %>%
  filter(n()>100) %>%
  ungroup()

test_recommender <- test_set %>%
  semi_join(train_recommender, by = "movieId") %>%
  semi_join(train_recommender, by = "userId")

#create recommender_matrix
recommender_matrix <- train_recommender %>%
  select(userId, movieId, rating) %>%
  spread(movieId, rating) %>%
  as.matrix()
#add row names by userId
rownames(recommender_matrix) <- recommender_matrix[,1]
recommender_matrix <- recommender_matrix[,-1]
movie_titles <- train_set %>%
  select(movieId, title) %>%
  distinct()
#add column names, title by movieId
colnames(recommender_matrix) <- with(movie_titles, title[match(colnames(recommender_matrix), movieId)])

#creat test_matrix
test_matrix <- test_recommender %>%
  select(userId, movieId, rating) %>%
```

```
  spread(movieId, rating) %>%
  as.matrix()
#add row names by userId
rownames(test_matrix) <- test_matrix[,1]
test_matrix <- test_matrix[,-1]
movie_titles_test <- test_set %>%
  select(movieId, title) %>%
  distinct()
#add column names, title by movieId
colnames(test_matrix) <- with(movie_titles_test, title[match(colnames(test_matrix), movieId)])

#coerce to realRatingMatrix as required by recommender
recommender_matrix <- as(recommender_matrix, "realRatingMatrix")
test_matrix <- as(test_matrix, "realRatingMatrix")

#check the class of the new matrix
class(recommender_matrix)
```

```
## [1] "realRatingMatrix"
## attr(,"package")
## [1] "recommenderlab"
```

```
class(test_matrix)
```

```
## [1] "realRatingMatrix"
## attr(,"package")
## [1] "recommenderlab"
```

After the data has been coerced into a realRatingMatrix, the recommender can be built using User-Based Collaborative Filtering 'UBCF'.

```
#create model based on user-user (UBCF) interactions
rec_model <-Recommender(recommender_matrix, method = "UBCF")
```

This model can then be analysed be predicting all the values in the test_matrix, replacing any values which are already present using the predict function, type = "ratingMatrix". The predictions can be evaluated, including RMSE, using the recommenderlab calcPredictionAccuracy function.

```
#complete the testMatrix based on the model
pred <- predict(rec_model, test_matrix, type="ratingMatrix")

#calculatepredictionaccuracy
calcPredictionAccuracy(pred, test_matrix)
```

```
##      RMSE       MSE       MAE
## 0.8416237 0.7083304 0.6474871
```

The calculated RMSE, 0.8416237, on the test set is much lower than that calculated by the previous models which shows the power of this approach. Unfortunately limited computational power limits the ability for this to be carried out on the entire edx dataset.