Predicting Movie Ratings

Dooruj Rambaccussing

2022-10-17

Overview

This project explores movies rating citations as part of the Harvard data science Capstone Course. The report first introduces the idea of movie rating citations using did analytics. Turn to take this project first the data set will be prepared for use. The project starts by first doing an exploratory data analysis. Based on the latter a machine learning algorithm will be used to predict movie ratings. The models are assessed using the root mean squared error and a model with mean, user ratings, the movie and the movie is found to be successful. Regularization marginally improves the root mean squared error.

Introduction to the Study

One of the most obvious use of machine learning techniques is for prediction. these include predictions of future outcome variables such as economic variables, climate change, business outcomes and demand for recreational activities such as movies. The concept of recommendation systems is used in many areas of businesses such as movie recommendations oh produce on Amazon or eBay. for the purpose of recommendations it is very useful to analyse big data sets and then predict behaviour base on the analysis.

For this project we will create a movie recommendation system using the MovieLens dataset. Both the original and validation set are well sized and contain important information such as the movie ID release year, the year the movie is rater, the user who rated the movie and the final rating.

Data

The data can be downloaded here. https://grouplens.org/datasets/movielens/latest/

The data preparation requires the following code:

```
# Note: this process could take a couple of minutes

# if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
# if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
# if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
rm(list=ls())
library(tidyverse)
library(caret)
library(data.table)

# MovieLens 10M dataset:
# https://grouplens.org/datasets/movielens/10m/
# http://files.grouplens.org/datasets/movielens/ml-10m.zip

dl <- tempfile()
download.file("https://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"))

#If Edx code does not work, use the following:

#ratings <- read.table(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")



# if using R 4.0 or later:
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(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") # if using R 3.5 or earlier, use `set.seed(1)`
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]

# Make sure userId and movieId in validation set are also in edx set
validation <- temp %>%
      semi_join(edx, by = "movieId") %>%
      semi_join(edx, by = "userId")

# Add rows removed from validation set back into edx set
removed <- anti_join(temp, validation)
edx <- rbind(edx, removed)

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

```
setwd("C:/Users/DRambaccussing/OneDrive - University of Dundee/HARVARDX/DOORUJRAMBACCUSSING")
rm(list = ls())
library(tidyverse)
library(caret)
library(data.table)
load("data1.RData")
```

Exploratory Data Analysis

The first step is to see whether our data has been adequately loaded, and understand the structure of the dataset. We perform some initial exploration of the data and set the class as a tibble:

```
class(edx)
```

```
## [1] "data.table" "data.frame"
```

```
which(is.na(edx))
```

```
## integer(0)
```

```
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)                     Comed~
##  2      1     185      5 838983525 Net, The (1995)                      Actio~
##  3      1     292      5 838983421 Outbreak (1995)                      Actio~
##  4      1     316      5 838983392 Stargate (1994)                      Actio~
##  5      1     329      5 838983392 Star Trek: Generations (1994)        Actio~
##  6      1     355      5 838984474 Flintstones, The (1994)              Child~
##  7      1     356      5 838983653 Forrest Gump (1994)                  Comed~
##  8      1     362      5 838984885 Jungle Book, The (1994)              Adven~
##  9      1     364      5 838983707 Lion King, The (1994)                Adven~
## 10      1     370      5 838984596 Naked Gun 33 1/3: The Final Insult (1~ Actio~
## # ... with 9,000,045 more rows
```

```
head(edx)
```

```
##    userId movieId rating timestamp                         title
## 1:      1     122      5 838985046              Boomerang (1992)
## 2:      1     185      5 838983525               Net, The (1995)
## 3:      1     292      5 838983421               Outbreak (1995)
## 4:      1     316      5 838983392               Stargate (1994)
## 5:      1     329      5 838983392 Star Trek: Generations (1994)
## 6:      1     355      5 838984474       Flintstones, The (1994)
##                          genres
## 1:               Comedy|Romance
## 2:         Action|Crime|Thriller
## 3:  Action|Drama|Sci-Fi|Thriller
## 4:       Action|Adventure|Sci-Fi
## 5: Action|Adventure|Drama|Sci-Fi
## 6:       Children|Comedy|Fantasy
```

```
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, 2, ~
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420, ~
## $ rating    <dbl> 5, 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, 83898~
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", "S~
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|Sci~
```

The initial data appears tidy, with no missing observations but it would seems that the time-stamps are integers. Moreover, the release years are in the title. So we next mutate the time stamp into a *Year* variable for both edx and validation datasets. We also create a new column for the release year.

```
edx <- edx %>% mutate(timestamp = as.POSIXct(timestamp, origin = "1970-01-01",
                                             tz = "UTC"))
edx$timestamp <- format(edx$timestamp, "%Y")
names(edx)[names(edx) == "timestamp"] <- "Year"
```

```
validation <- validation %>% mutate(timestamp = as.POSIXct(timestamp, origin = "1970-01-01",
                                                           tz = "UTC"))
validation$timestamp <- format(validation$timestamp, "%Y")
names(edx)[names(validation) == "timestamp"] <- "Year"
## Release year
Releaseyear =as.numeric(str_sub(edx$title, start =-5, end =-2))
edx = edx %>% mutate(Release_year = Releaseyear)
Releaseyear =as.numeric(str_sub(validation$title, start =-5, end =-2))
validation = validation %>% mutate(Release_year = Releaseyear)
```

In this report we shall also consider two variables which are behavioural in nature namely the movie age and the number of characters in the title. Therefore two new variables *movie age.* and *char_length.*

```
edx = edx %>% mutate(MovieAge = 2022 - Release_year)
validation = validation %>% mutate(MovieAge = 2022 - Release_year)

edx = edx %>% mutate(char_length = nchar(title))
validation = validation %>% mutate(char_length = nchar(title))
```

Data description and summary plots

The following codes perform an initial exploration of the data. We compute the number of users, movies, and table the rating and year of rating distribution.

```
#Number of Users
length(unique(edx$userId))
```

```
## [1] 69878
```

```
#Number of movies
length(unique(edx$movieId))
```

```
## [1] 10677
```

```
# Table of ratings and sample length
table(edx$rating)
```

```
##
##     0.5       1     1.5       2     2.5       3     3.5       4     4.5       5
##   85374  345679  106426  711422  333010 2121240  791624 2588430  526736 1390114
```

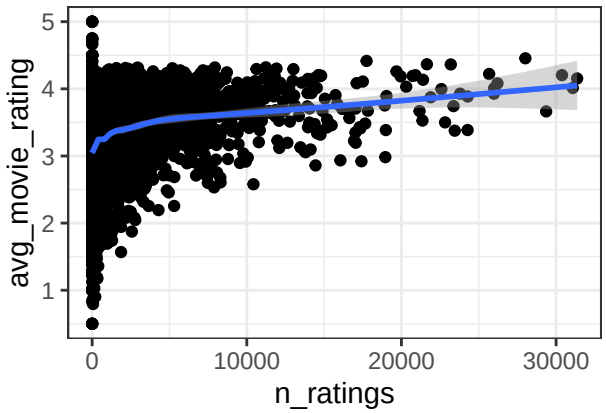
```
table(edx$Year)
```

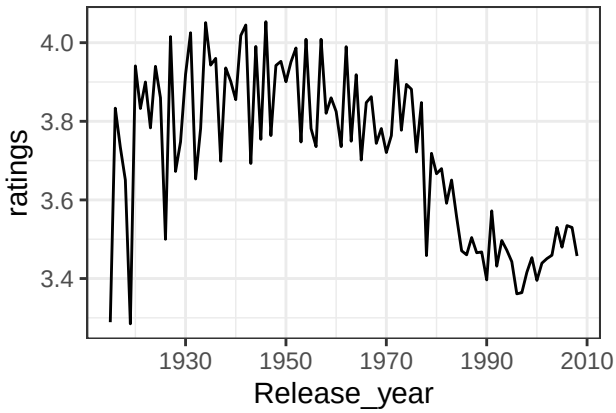
```
##
##    1995    1996    1997    1998    1999    2000    2001    2002    2003    2004
##       2  942772  414101  181634  709893 1144349  683355  524959  619938  691429
##    2005    2006    2007    2008    2009
## 1059277  689315  629168  696740   13123
```

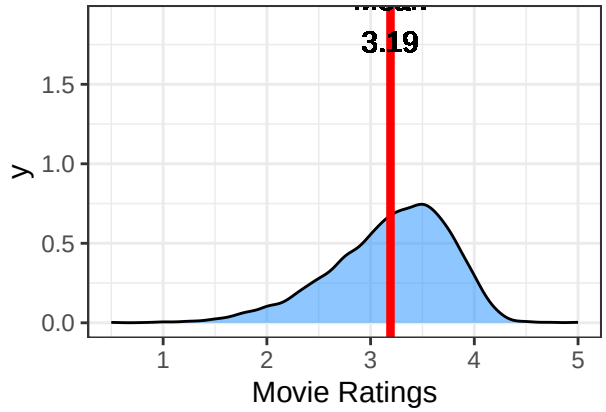
The edx dataset has 10677 unique raters, with 69878 unique movies which were rated from 1995 until 2009. A basic analysis of the ratings score show that users are most likely to rate movies as integers instead of fractions (such as 2.5 or 3.5).

```
movie = edx %>%
  group_by(movieId,title) %>%
  summarise(n_ratings = n(),
            avg_movie_rating = mean(rating)
            )
```

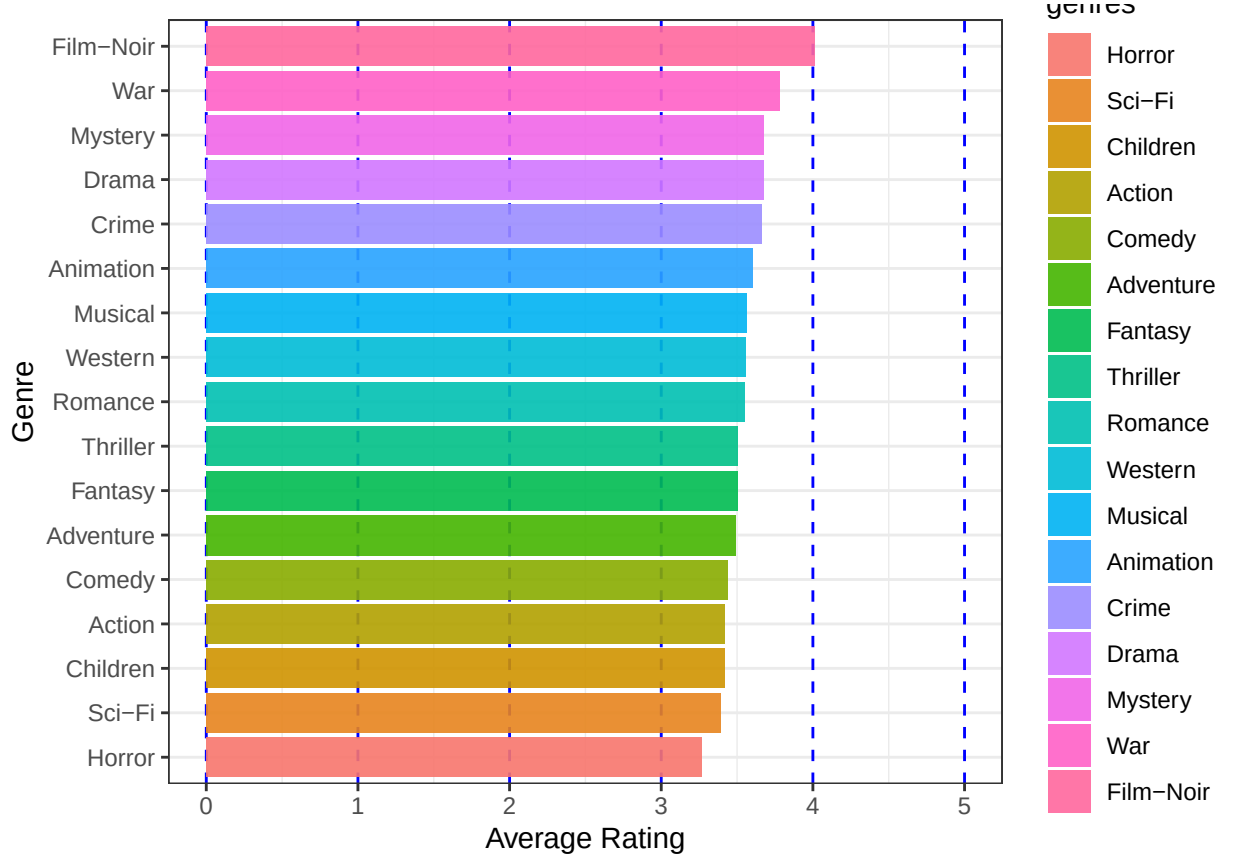
```
## Scatterplot of number of times rated and average ratings
A=movie %>%
  ggplot(aes(n_ratings,avg_movie_rating))+
  geom_point(aes())+
  geom_smooth()+
  theme_bw()


timeseries = edx %>%
  group_by(Release_year) %>%
  summarise(ratings = mean(rating), count=n())

B=ggplot(data=timeseries, aes(x=Release_year, y=ratings, group=1)) +
  geom_line(linetype = "solid")+
  theme(axis.text.x = element_text(angle = 90, hjust = 0.2))+
  theme_bw()



density = edx %>%
  group_by(movieId) %>%
  summarise(ratings = mean(rating))
mean_rat = mean(density$ratings)

C= ggplot(density, aes(x=ratings))+
  geom_density(fill="dodgerblue", alpha=0.5)+
  labs(x="Movie Ratings")+
  geom_vline(xintercept=mean_rat, size=1.5, color="red")+
  geom_text(aes(x=3.19, label=paste0("Mean\n",3.19), y=1.9))+
  theme_bw()

if(!require(gridExtra)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
library("gridExtra")
grid.arrange(A, B, C, ncol = 2, nrow = 2)
```







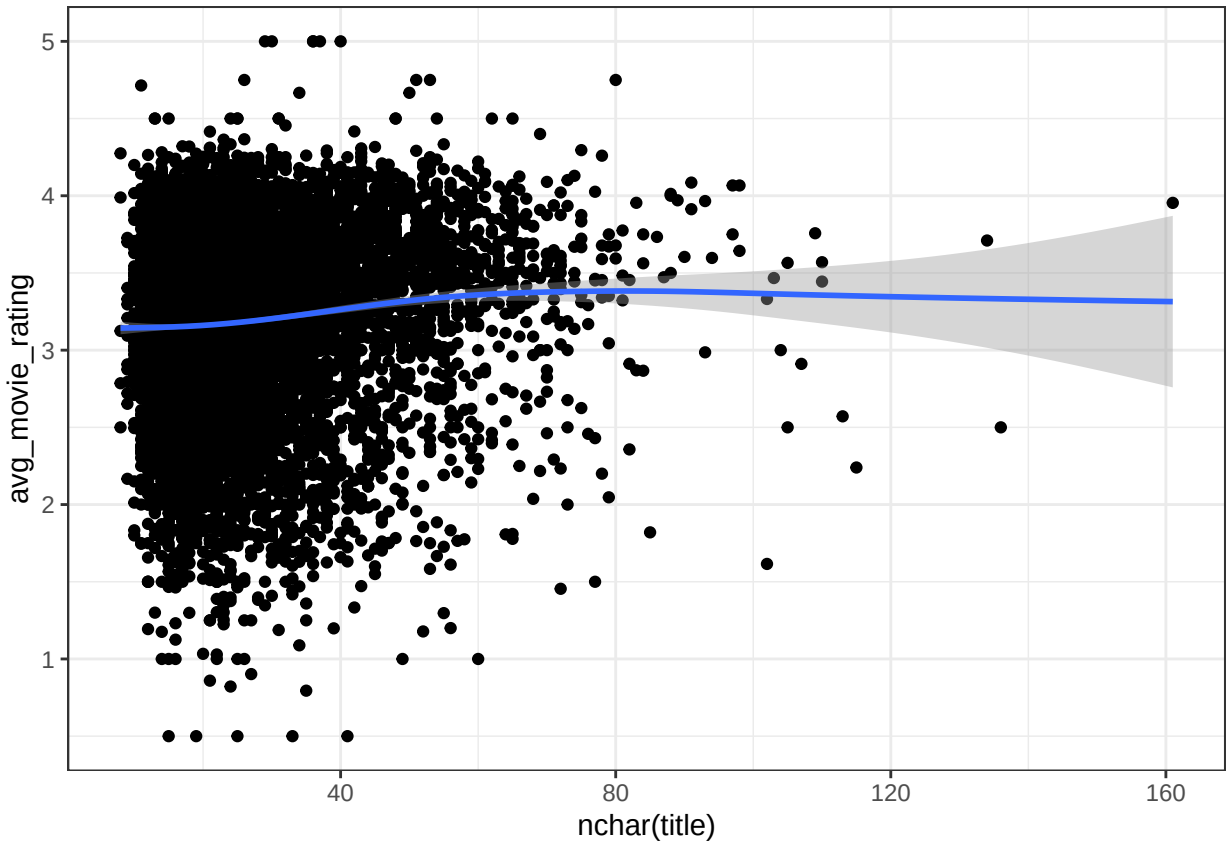

```
genres_df <- edx %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarise(number = n(), averageratings = mean(rating))

## Here we discard genres where there were less than 100000 as the barplot would return fairly big.
genres_df  %>%
  filter(number > 100000) %>%
  mutate(genres = fct_reorder(genres, averageratings)) %>%
  ggplot(aes(x = averageratings, y = genres)) +
  geom_col(aes(fill=genres), alpha = 0.9) +
  labs(y = "Genre",
       x = "Average Rating") +
  theme_bw() +
  scale_x_continuous(limits = c(0, 5),
                     breaks = 0:5,
                     labels = 0:5) +
  theme(panel.grid.major.x = element_line(linetype = "dashed", color = "blue"))
```




```
####
```

An examination of the density plot show that the distribution of movie ratings is slightly skewed with a mean of 3.19. The time series average ratings show that movies which were released earlier in time have higher ratings generally. Classics, such as movies released between 1930 and 1970, are rated higher than the most recent releases. The scatter plot between movie ratings per movie and the number of ratings seems to be only slightly correlated, where movies with higher ratings seem to be slightly better rated. When the genre of the movie was considered, it turned out that some types of movies, such as Film Noir and War were rated higher than Mystery movies.

```
## `geom_smooth()` using method = 'gam' and formula = 'y ~ s(x, bs = "cs")'
```




```
## [1] 0.09409092
```

The character length of the title was found to quite low but positive, which perhaps suggest that there could be an evidence of positive ratings for movies with more characters.

Methods

To develop a classifier system for the movies, we consider the following unested model as being representative of the movie rating:

$$ratings_i = \alpha + b_i + b_u + b_g + b_t + b_c + \epsilon_i$$

where alpha is the mean of the total movie ratings, b_i is the bias due to the individual movie, b_u is the user bias, b_g is the genre bias, b_t is the movie age bias, and b_c is the character length bias.

Our model shall sequentially consider adding these parameters together and test for its accuracy. Many accuracy measures can be chosen for testig the above model such as the Mean Squared Error, Theil's U and the Mean absolute error. This paper uses the root mean squared (RMSE) as measure given its wide use. The RMSE measures the differences between values predicted by a model and the values observed. The lower the RMSE, the better the model. The RMSE is however sensitive to outliers in forecasts. The RMSE is computed as follows:

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

Partitioning data into Training and Test Sets

Since the model contains five components namely: movie, user, genre, release_year and character, the edx dataset is sliced into a training and a test set. The partitioning is helped by the caret package:

```
##partitioning the dataset into test and training samples.
test_index = createDataPartition(y = edx$rating, times= 1, p = 0.2,
                                 list = F
                                 )
test_set <- edx %>% slice(test_index)
train_set <- edx %>% slice(-test_index)
## free memory
rm(density, densit, genres_df, Genres, movie, test_index, timeseries)
```

We consider an 80-20 split in the training to test sample. This is because our dataset is large enough to consider a big testing sample, which ensures a more reliable RMSE asymptotically. The models will first be estimated using the training sample and then the validation will be performed on the test set.

The models to be estimated are as follows:

Mean Only :

$$ratings_i = \alpha$$

Mean with movie effect:

$$ratings_i = \alpha + b_i$$

Mean with movie and user effect:

$$ratings_i = \alpha + b_i + b_u$$

mean with movie, user and genre effect

$$ratings_i = \alpha + b_i + b_u + b_g$$

Mean with movie, user, genre and time effect

$$ratings_i = \alpha + b_i + b_u + b_g + b_t$$

Mean with movie, user, genre, time and character effects.

These will in turn be assessed insample and the corresponding parameters will be used for out of sample forecasts.

Results

The insample parameters can be estimated:

```
#### solving Parameters of alpha (mean), bi(movie) and bu (user),
### bt (release year), bg(genres) and bc(characters in titles)
alpha <- mean(train_set$rating)

bi <- train_set %>%
  group_by(movieId) %>%
  summarize(b_i = mean(rating - alpha))

bu <- train_set %>%
```

```
  left_join(bi, by = "movieId") %>%
  group_by(userId) %>%
  summarize(b_u = mean(rating - alpha - b_i))

bg <- train_set %>%
  left_join(bu, by = "userId") %>%
  left_join(bi, by = "movieId") %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - alpha - b_i - b_u))

bt <- train_set %>%
  left_join(bu, by = "userId") %>%
  left_join(bi, by = "movieId") %>%
  left_join(bg, by = "genres") %>%
  group_by(Release_year) %>%
  summarize(b_t = mean(rating - alpha - b_i - b_u -b_g))

bc <- train_set %>%
  left_join(bu, by = "userId") %>%
  left_join(bi, by = "movieId") %>%
  left_join(bg, by = "genres") %>%
  left_join(bt, by = "Release_year") %>%
  group_by(char_length) %>%
  summarize(b_c = mean(rating - alpha - b_i - b_u -b_g -b_t))
```

The test sample forecasts are estimated using:

```
## Predictions

n_rmse <- RMSE(test_set$rating, alpha)
n_rmse
```

```
## [1] 1.060708
```

```
rmse_results <- data_frame(method = "Average movie rating", RMSE = n_rmse)
```

```
## Warning: `data_frame()` was deprecated in tibble 1.1.0.
## i Please use `tibble()` instead.
```

```
pred_bi <- alpha + test_set %>%
  left_join(bi, by = "movieId") %>%
  pull(b_i)

model1_rmse <- RMSE(pred = pred_bi,
                    obs = test_set$rating,
                    na.rm = TRUE)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie effect ",
                                     RMSE = model1_rmse ))

pred_bu <- test_set %>%
  left_join(bi, by = "movieId") %>%
  left_join(bu, by = "userId") %>%
  mutate(pred_bu = alpha + b_i + b_u) %>%
  pull(pred_bu)
```

```
model2_rmse <- RMSE(pred = pred_bu,
                    obs = test_set$rating,
                    na.rm = TRUE)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie and User effect",
                                     RMSE = model2_rmse))
# Free some space
rm(pred_bi, pred_bu)

pred_bg <- test_set %>%
  left_join(bi, by = "movieId") %>%
  left_join(bu, by = "userId") %>%
  left_join(bg, by = "genres") %>%
  mutate(pred_bg = alpha + b_i + b_u + b_g) %>%
  pull(pred_bg)

model3_rmse <- RMSE(pred = pred_bg,
                    obs = test_set$rating,
                    na.rm = TRUE)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie, User and Genre effect",
                                     RMSE = model3_rmse))

pred_bt <- test_set %>%
  left_join(bu, by = "userId") %>%
  left_join(bi, by = "movieId") %>%
  left_join(bg, by = "genres") %>%
  left_join(bt, by = "Release_year") %>%
  mutate(pred_bt = alpha + b_i + b_u + b_g+ b_t) %>%
  pull(pred_bt)

model4_rmse <- RMSE(pred = pred_bt,
                    obs = test_set$rating,
                    na.rm = TRUE)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie, User, Genre and Release date effect",
                                     RMSE = model4_rmse))


pred_bc <- test_set %>%
  left_join(bu, by = "userId") %>%
  left_join(bi, by = "movieId") %>%
  left_join(bg, by = "genres") %>%
  left_join(bt, by = "Release_year") %>%
  left_join(bc, by = "char_length") %>%
  mutate(pred_bc = alpha + b_i + b_u + b_g+ b_t +b_c) %>%
  pull(pred_bc)

model5_rmse <- RMSE(pred = pred_bc,
                      obs = test_set$rating,
```

```
                      na.rm = TRUE)

rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Movie, User, Genre, Release date and Character length effe
                                     RMSE = model4_rmse))
rmse_results %>% knitr::kable()
```

method	RMSE
Average movie rating	1.0607079
Movie effect	0.9437144
Movie and User effect	0.8661625
Movie, User and Genre effect	0.8658242
Movie, User, Genre and Release date effect	0.8656468
Movie, User, Genre, Release date and Character length effect	0.8656468

The root mean square error results are interesting. The centered mean (naive) root mean squared error is quite high (1.06) for the model does not make a distinction across each individual movie. Obviously if distribution of ratings were very centered around 3.19, we may expect a lower rmse. By adding on the movie effect, The model improves significantly as the the RMSE goes down to 0.943. By taking into account that the model is rated by individuals who have their own attitudes towards rating movies, the RMSE falls further to 0.866. Based on the table of RMSE, it would seem that these two constitute account for most of the heterogeneity in the rating systems. When adding on the genre, again the RMSE falls further to 0.865, which is somehow smaller than before. However adding on the release year and the title length can only lead to minute improvements in RMSE.

Validation sample

Based on our earlier findings on the test sets, it would seem that a model of the naive mean with movie, user and genre would be appropriate. Since the addition of time and character would lead to only small improvements in RMSE, it is likely that these models would also lead to a high variance. We first replicate the predictions for the first four models.

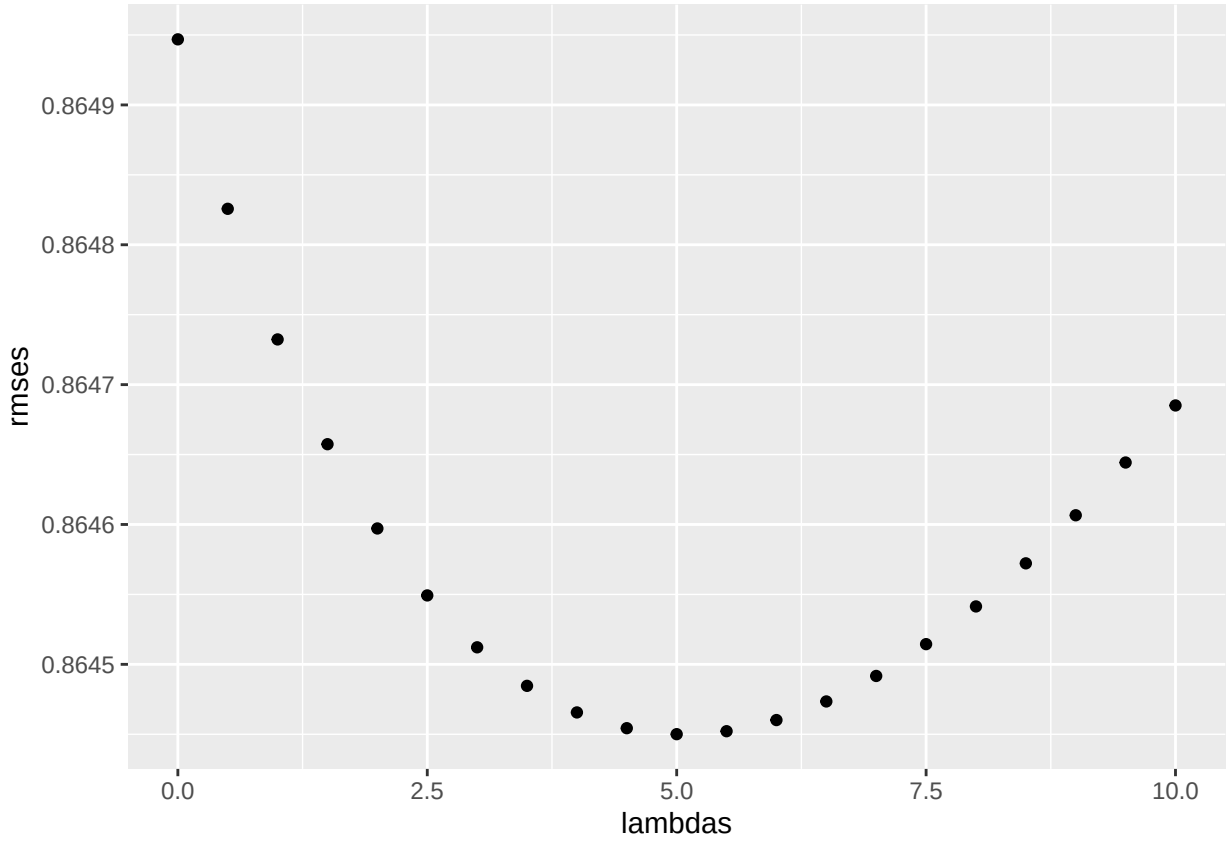
method	RMSE
Average movie rating model	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488
Movie, user and Genre effect model	0.8649469

Regularization

We also consider regularization techniques for the chosen model of mean with genre, user and movie effects. The use of a regularized model is motivated by the fact that b_i, b_u and b_g are led by movies and genres with few ratings and where users rated only a small number of movies. Regularization adds on a penalty for each of those parameter by computing a value of λ which will shrink the parameters when a small number of ratings are involved.

```
##########################Regularisation###
# lambda is a tuning parameter
# Use cross-validation to choose it.
lambdas <- seq(0, 10, 0.5)
```

```r
# For each lambda,find b_i & b_u, followed by rating prediction & testing
# note:the below code could take some time
rmses <- sapply(lambdas, function(l){

  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)

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

# Plot rmses vs lambdas for optimal lambda
qplot(lambdas, rmses)
```




```
#Determine which lambda minimizes the RMSE
lambda <- lambdas[which.min(rmses)]
```

The lambda which minimises the RMSE = 5. The table with the RMSE for the four models and including the regularized model are as follows:

method	RMSE
Average movie rating model	1.0612018
Movie effect model	0.9439087
Movie and user effect model	0.8653488
Movie, user and Genre effect model	0.8649469
Regularized movie,user and genre effect	0.8644501

The introduction of the regularized model with the genre, user and movie turns out to be marginally better. Without regularization, the root mean squared error is equal to 0.8649469 where as with regularized model it is equal to 0.8644501.

Conclusion

In a nutshell, this report builds a machine learning algorithm to help predict ratings using the MovieLens dataset. The regularized model with the movie, user and genre turns out to be the best model for forecasting movie ratings as it has the lowest RMSE value (0.864). However it should be noted that the regularized version is not far off from the unregularized model which is 0.865. It is worth noting that there could be other machine learning algorithms which have been tested but yet could achieve lower RMSE.