# Movielens exercise

**Author:** Alberto Guijarro Rodriguez

**Date:** 2021-05-20

**Loc:** England

# Executive Summary

Built algorithm capable of accurately predicting user rating score on film. The **target** for the model to be considered efficient enough is to be able to reach a **RMSE on the validation set under 0.86490**. Due to the size of the dataset to be processed and the constraints of home desktop computers the utilization of regular machine learning models is not recommended, as an example, conducting one hot econding on filmId (10,677 different movies in training set) and using an integer of 1 byte to store the information will require about 90GiB worth of memory, which is far outside what a desktop machine usually has in its RAM.

Developed algorithm was able by building a set of mapping dictionaries with expected rating values per:

1. Film
2. User
3. User and Genre

Was capable of **achieving a RMSE of 0.85619, which is under 0.86490**, to achieve this, aside of the mapping dictionaries a regularization approach was also put into place to account for some existing overfitting problems.

# Dataset description

Below we can see the output of the "str" function, from the R language, which provides an overview of the features, or columns available in our dataset. The target value for our function is the "rating" columns which contains a floating point number which ranges from 0.5 up to 5 in increments of 0.5. The other available columns can be described as follows:

- userId: int, unique number assigned to each user, total unique users in training set 69878
- movieId: int, unique number assigned to each movie, total unique movies in training set 10677
- genres: chr, string describing the type of genre each film belongs to, total unique genres in training set 797
- timestamp: int, seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970
- title: chr, text describing the movieId, we can extract the release date of the movie out of the "()" from the title field

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.0.4
```

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

```
## v ggplot2 3.3.2     v purrr   0.3.4
## v tibble  3.0.4     v dplyr   1.0.5
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0
```

```
## Warning: package 'dplyr' was built under R version 4.0.4
```

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

```
library(caret)
```

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

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

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

```
library(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
```

```
# 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")

# # if using R 3.6 or earlier:
# movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
#                                            title = as.character(title),
#                                            genres = as.character(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)`
```

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

```
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)

# edx<-read.csv('./edx.csv')
# validation<-read.csv('./validation.csv')

print("EDX Dataset dimensions")
```

```
## [1] "EDX Dataset dimensions"
```

```
dim(edx)
```

```
## [1] 9000055       6
```

```
print("EDX Dataset Structure")
```

```
## [1] "EDX Dataset Structure"
```

```
str(edx)
```

```
## Classes 'data.table' and 'data.frame':   9000055 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 292 316 329 355 356 362 364 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983421 838983392 838983392 838984474 838983653 838984885 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Outbreak (1995)" "Stargate (1994)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Action|Drama|Sci-Fi|Thriller" "Action|Ad
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
print(paste("EDX Total number unique movieId: ",length(unique(edx$movieId))))
```

```
## [1] "EDX Total number unique movieId:  10677"
```

```
print(paste("EDX Total number unique userId: ",length(unique(edx$userId))))
```

```
## [1] "EDX Total number unique userId:  69878"
```

```
print(paste("EDX Total number unique genres: ",length(unique(edx$genres))))
```

```
## [1] "EDX Total number unique genres:  797"
```

Added field in order to build our mapping dictionaries: - guk: chr, concatenated value of userId and genre, to account for the preference each user might have for each specific genre

```
edx<-edx%>%mutate(guk = paste(userId,genres))
validation<-validation%>%mutate(guk = paste(userId,genres))
```

# Method & Analysis

Initially we trialed our approach against the entire dataset, assessing our improvements each step of the way, however, before moving into the different analysis steps, we will first build the required support functions to conduct the analysis:

**Support Functions**

```
RMSE <- function(actuals,forecast){
  sqrt(mean((actuals-forecast)**2))
}
```

Initially we built a high level model which factored in the average score each user usually gives to any given film and the average score each film migh usually get, to do this, instead of building a linear regressor, given the size of the dataset, we built a **movies_coefs** variable which is a map to later add the calculated effect to each movie on the validation set.

```
data_model<-edx
val_data_model<-validation

base_rating <- mean(data_model$rating)

data_model <- data_model %>% mutate(
  br=base_rating,
  fcast_rating=br,
  resid=rating-fcast_rating
)

val_data_model <- val_data_model %>% mutate(
  br=base_rating,
  fcast_rating=br,
```

```
  resid=rating-fcast_rating
)


# First we will calculate the usual score given to each movie
movies_coefs <-  data_model %>% as_tibble() %>% group_by(movieId) %>%
  summarise(mf = mean(resid)) %>%
  select(movieId, mf)


#data_model<-data_model%>%mutate(br=base_rating)
data_model<-merge(data_model, movies_coefs,
                  by.x = c("movieId"),
                  by.y = c("movieId"),
                  all.x = TRUE,
                  all.y = FALSE) %>%
  mutate(fcast_rating=br+mf, resid=rating-fcast_rating)


val_data_model<-val_data_model%>%mutate(br=base_rating)
val_data_model<-merge(val_data_model, movies_coefs,
                  by.x = c("movieId"),
                  by.y = c("movieId"),
                  all.x = TRUE,
                  all.y = FALSE) %>%
  mutate(fcast_rating=br+mf, resid=rating-fcast_rating)

RMSE(data_model$rating, data_model$fcast_rating)
```

```
## [1] 0.9423475
```

```
RMSE(val_data_model$rating, val_data_model$fcast_rating)
```

```
## [1] 0.9439087
```

The RMSE at this stage was:

- EDX dataset: 0.9423475449724072
- Validation: 0.9439086628063089

The next thing to consider will be to include the user effect, which could be understood as the usual score each user might give to each film, for this purpose the variable **user_coefs** has been created, which contains the average effect each user usually has when scoring a movie.

```
# user_coefs
# Second we will calculate the usual score given by each user
user_coefs <-  data_model %>% group_by(userId) %>%
  summarise(uf = mean(resid)) %>%
  select(userId, uf)
```

```
data_model<-merge(data_model, user_coefs,
                  by.x = c("userId"),
                  by.y = c("userId"),
                  all.x = TRUE,
                  all.y = FALSE) %>%
  mutate(fcast_rating=br+mf+uf, resid=rating-fcast_rating)


val_data_model<-merge(val_data_model, user_coefs,
                      by.x = c("userId"),
                      by.y = c("userId"),
                      all.x = TRUE,
                      all.y = FALSE) %>%
  mutate(fcast_rating=br+mf+uf, resid=rating-fcast_rating)

RMSE(data_model$rating, data_model$fcast_rating)
```

```
## [1] 0.8567039
```

```
RMSE(val_data_model$rating, val_data_model$fcast_rating)
```

```
## [1] 0.8653488
```

The RMSE at this stage was:

- EDX dataset: 0.8567038923601745
- Validation: 0.8653488245773153

Lastly we will include the user-genre interaction effect, trying to cater for the fact, that each user will have different tastes regarding each genre type. For this purpose, we build the **guk_coefs** variable, given we might find some instance where the combination user genre might not exist in our validation set, we will fill those values with a 0

```
# First we will calculate the usual score given by each user to each genre
guk_coefs <-  data_model %>% group_by(guk) %>%
  summarise(gf = mean(resid)) %>%
  select(guk, gf)


data_model<-merge(data_model, guk_coefs,
                  by.x = c("guk"),
                  by.y = c("guk"),
                  all.x = TRUE,
                  all.y = FALSE) %>%
  mutate(fcast_rating=br+mf+uf+gf, resid=rating-fcast_rating)

val_data_model<-merge(val_data_model, guk_coefs,
                      by.x = c("guk"),
                      by.y = c("guk"),
                      all.x = TRUE,
                      all.y = FALSE)
```

```
# Fill Missing Values
val_data_model$gf[is.na(val_data_model$gf)]<-0
val_data_model<-val_data_model %>% mutate(fcast_rating=br+mf+uf+gf, resid=rating-fcast_rating)

RMSE(data_model$rating, data_model$fcast_rating)
```

```
## [1] 0.5840494
```

```
RMSE(val_data_model$rating, val_data_model$fcast_rating)
```

```
## [1] 0.9182867
```

The RMSE at this stage was:

- EDX dataset: 0.5840493997405306
- Validation: 0.9182866579994383

The change in RMSE over the validation set, shows a clear case of "overfitting", where we are correcting very well for our training set, but the adjustment is not proving correct with the validation set.

## Understanding the Overfitting Problem

In order to understand what is happening, we will split out edx training set, into two subsets Train and Test, and use these subsetsto try to understand what is happening with the validation set, **following the spirit of the exercise which is "not to use the validation set only than to measure RMSE results".**

```
set.seed(42, sample.kind="Rounding")
```

```
## Warning in set.seed(42, sample.kind = "Rounding"): non-uniform 'Rounding'
## sampler used
```

```
test_index <- createDataPartition(edx$movieId, p=0.3, list = FALSE)
edx_train<-edx[-test_index,] # train dataset
edx_test<-edx[test_index,] # test dataset
```

If we were to retrain our model but this time using this subsets of training/test, we could **check the largest differences in resid instances**, and drive conclusions from there.

```
data_model<-edx_train
val_data_model<-edx_test


# Given we might end up with missing values
# We will have a model which produces a "base rating"
# And then we will include the other effects
base_rating <- mean(data_model$rating)

data_model <- data_model %>% mutate(
  br=base_rating,
```

```
  fcast_rating=br,
  resid=rating-fcast_rating
)

val_data_model <- val_data_model %>% mutate(
  br=base_rating,
  fcast_rating=br,
  resid=rating-fcast_rating
)

# RMSE(data_model$rating, data_model$fcast_rating)
# RMSE(val_data_model$rating, val_data_model$fcast_rating)


# movies_coefs
# First we will calculate the usual score given to each movie
movies_coefs <-  data_model %>% as_tibble() %>% group_by(movieId) %>%
  summarise(mf = mean(resid)) %>%
  select(movieId, mf)


#data_model<-data_model%>%mutate(br=base_rating)
data_model<-merge(data_model, movies_coefs,
                  by.x = c("movieId"),
                  by.y = c("movieId"),
                  all.x = TRUE,
                  all.y = FALSE) %>%
  mutate(fcast_rating=br+mf, resid=rating-fcast_rating)


val_data_model<-val_data_model%>%mutate(br=base_rating)
val_data_model<-merge(val_data_model, movies_coefs,
                      by.x = c("movieId"),
                      by.y = c("movieId"),
                      all.x = TRUE,
                      all.y = FALSE)

val_data_model$mf[is.na(val_data_model$mf)]<-0
val_data_model<-val_data_model%>%mutate(fcast_rating=br+mf, resid=rating-fcast_rating)

# RMSE(data_model$rating, data_model$fcast_rating)
# RMSE(val_data_model$rating, val_data_model$fcast_rating)



# user_coefs
# Second we will calculate the usual score given by each user
user_coefs <-  data_model %>% group_by(userId) %>%
  summarise(uf = mean(resid)) %>%
  select(userId, uf)


data_model<-merge(data_model, user_coefs,
```

```
                   by.x = c("userId"),
                   by.y = c("userId"),
                   all.x = TRUE,
                   all.y = FALSE) %>%
  mutate(fcast_rating=br+mf+uf, resid=rating-fcast_rating)


val_data_model<-merge(val_data_model, user_coefs,
                      by.x = c("userId"),
                      by.y = c("userId"),
                      all.x = TRUE,
                      all.y = FALSE)

val_data_model$uf[is.na(val_data_model$uf)]<-0
val_data_model<-val_data_model%>%mutate(fcast_rating=br+mf+uf, resid=rating-fcast_rating)

# RMSE(data_model$rating, data_model$fcast_rating)
# RMSE(val_data_model$rating, val_data_model$fcast_rating)


# genre_coefs
# First we will calculate the usual score given by each user to each genre
guk_coefs <-  data_model %>% group_by(guk) %>%
  summarise(gf = mean(resid)) %>%
  select(guk, gf)


data_model<-merge(data_model, guk_coefs,
                  by.x = c("guk"),
                  by.y = c("guk"),
                  all.x = TRUE,
                  all.y = FALSE) %>%
  mutate(fcast_rating=br+mf+uf+gf, resid=rating-fcast_rating)

val_data_model<-merge(val_data_model, guk_coefs,
                      by.x = c("guk"),
                      by.y = c("guk"),
                      all.x = TRUE,
                      all.y = FALSE)

# Fill Missing Values
val_data_model$gf[is.na(val_data_model$gf)]<-0
val_data_model<-val_data_model %>% mutate(fcast_rating=br+mf+uf+gf, resid=rating-fcast_rating)

RMSE(data_model$rating, data_model$fcast_rating)
```

```
## [1] 0.555703
```

```
RMSE(val_data_model$rating, val_data_model$fcast_rating)
```

```
## [1] 0.924556
```

The RMSE at this stage was:

- EDX dataset: 0.555703
- Validation: 0.924556

Which shows a similar problem as before, where adding the extra granualrity of the user and genre effect improves score against our train set, but damages our test set.

If we were to look at the largest differences on the test set, we can appreciate below how user 37651 has a residual of 6.122, so if we were to, consider the scores this user usually gives to Comedy|Drama films, we could appreciate if there is a problem of overfitting,

```
val_data_model %>%
  arrange(desc(resid))%>%
  head()
```

```
##                              guk userId movieId rating  timestamp
## 1:            37651 Comedy|Drama  37651    4254    4.5 1202070920
## 2:          38900 Comedy|Musical  38900    1760    4.5 1069245437
## 3:      21267 Action|Crime|Drama  21267    6587    5.0 1060799787
## 4:  5046 Action|Adventure|Sci-Fi   5046    4638    5.0 1002125897
## 5: 65546 Action|Adventure|Sci-Fi  65546    4638    5.0 1003607431
## 6:           7886 Comedy|Romance   7886    2597    5.0  987688583
##                                    title                 genres       br
## 1: Crocodile Dundee in Los Angeles (2001)           Comedy|Drama 3.512737
## 2:                      Spice World (1997)         Comedy|Musical 3.512737
## 3:                           Gigli (2003)      Action|Crime|Drama 3.512737
## 4:               Jurassic Park III (2001) Action|Adventure|Sci-Fi 3.512737
## 5:               Jurassic Park III (2001) Action|Adventure|Sci-Fi 3.512737
## 6:                    Lost & Found (1999)          Comedy|Romance 3.512737
##    fcast_rating    resid         mf         uf        gf
## 1:   -1.6224307 6.122431 -1.3322635  1.0245746 -4.827479
## 2:   -1.6176545 6.117655 -1.7759528  0.5494362 -3.903875
## 3:   -1.0100987 6.010099 -2.3384250 -1.1621400 -1.022271
## 4:   -0.6065268 5.606527 -0.8940888 -0.5434049 -2.681770
## 5:   -0.5764696 5.576470 -0.8940888  0.8242459 -4.019364
## 6:   -0.5303822 5.530382 -1.0443461 -1.2717682 -1.727005
```

Below we can see, that in our training set, this user only has one record regarding the Comedy|Drama genre, which is likely to skew the results towards this single score, instead than provide the general taste for the user for Comedy|Dramas.

```
data_model %>% filter(guk=="37651 Comedy|Drama") %>%
  head()
```

```
##                   guk userId movieId rating  timestamp
## 1: 37651 Comedy|Drama  37651    1193    0.5 1202076524
##                                     title       genres       br fcast_rating
## 1: One Flew Over the Cuckoo's Nest (1975) Comedy|Drama 3.512737          0.5
##    resid        mf       uf        gf
## 1:     0 0.7901672 1.024575 -4.827479
```

### Regularization

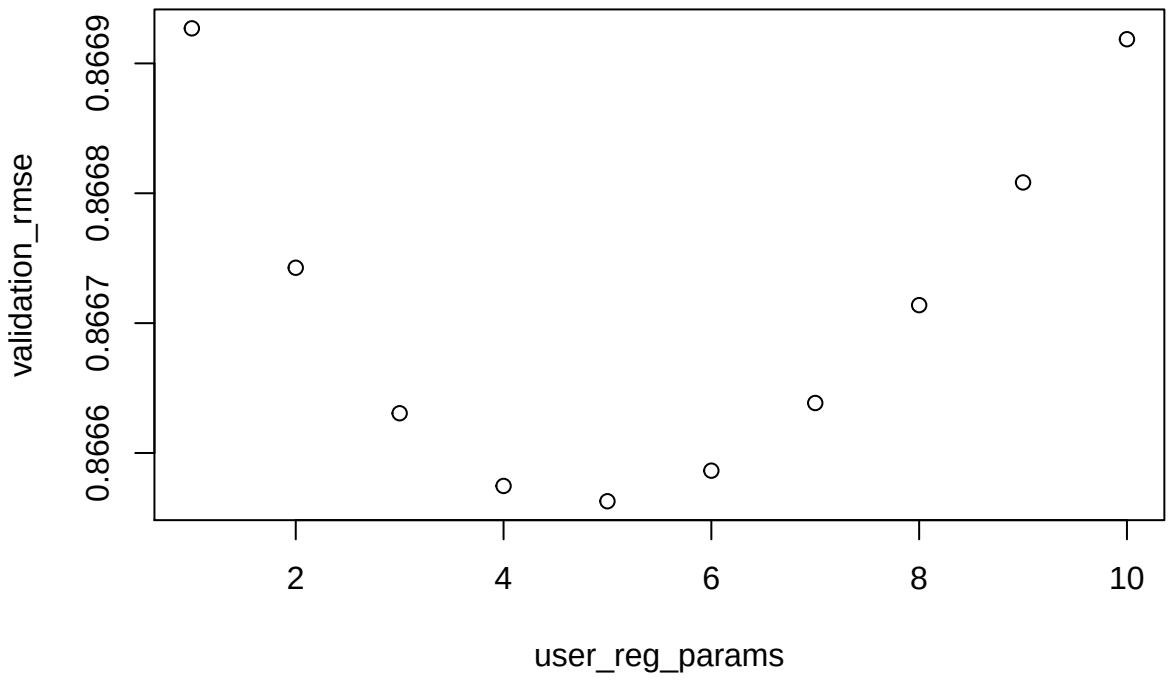
In order to overcome this overfitting problem we will apply regularization, but instead of just applying regularization at user|genre level we will also apply regularization at user level, in case a user just began

to provide movie scores and we still do not have a large enough sample for him/her to provide an accurate measure.

Function below, tries several posible values for the regularization parameter for user and plots them at the end, what we can see is that the optimum value to minimise the RMSE for the test set is to use a user regularization parameter of 5.

```
# Loop
user_reg_params = c(1:10)
validation_rmse<-list()

for (user_reg_param in user_reg_params) {

  ### User Reg Parameter
  data_model<-edx_train
  val_data_model<-edx_test

  base_rating <- mean(data_model$rating)

  data_model <- data_model %>% mutate(
    br=base_rating,
    fcast_rating=br,
    resid=rating-fcast_rating
  )

  val_data_model <- val_data_model %>% mutate(
    br=base_rating,
    fcast_rating=br,
    resid=rating-fcast_rating
  )

  # This will remain constant
  movies_coefs <-  data_model %>% as_tibble() %>% group_by(movieId) %>%
    summarise(mf = mean(resid)) %>%
    select(movieId, mf)

  data_model<-merge(data_model, movies_coefs,
                    by.x = c("movieId"),
                    by.y = c("movieId"),
                    all.x = TRUE,
                    all.y = FALSE) %>%
    mutate(fcast_rating=br+mf, resid=rating-fcast_rating)


  val_data_model<-val_data_model%>%mutate(br=base_rating)
  val_data_model<-merge(val_data_model, movies_coefs,
                        by.x = c("movieId"),
                        by.y = c("movieId"),
                        all.x = TRUE,
                        all.y = FALSE)

  val_data_model$mf[is.na(val_data_model$mf)]<-0
  val_data_model<-val_data_model%>%mutate(fcast_rating=br+mf, resid=rating-fcast_rating)
```

```
  user_coefs <-  data_model %>% group_by(userId) %>%
    summarise(suf = sum(resid), n=n(), uf = suf/(user_reg_param + n)) %>%
    select(userId, uf)

  data_model<-merge(data_model, user_coefs,
                    by.x = c("userId"),
                    by.y = c("userId"),
                    all.x = TRUE,
                    all.y = FALSE) %>%
    mutate(fcast_rating=br+mf+uf, resid=rating-fcast_rating)


  val_data_model<-merge(val_data_model, user_coefs,
                        by.x = c("userId"),
                        by.y = c("userId"),
                        all.x = TRUE,
                        all.y = FALSE)

  val_data_model$uf[is.na(val_data_model$uf)]<-0
  val_data_model<-val_data_model%>%mutate(fcast_rating=br+mf+uf, resid=rating-fcast_rating)

  # RMSE(data_model$rating, data_model$fcast_rating)
  # RMSE(val_data_model$rating, val_data_model$fcast_rating)
  validation_rmse[[user_reg_param]]<-RMSE(val_data_model$rating, val_data_model$fcast_rating)

}

plot(user_reg_params,validation_rmse)
```

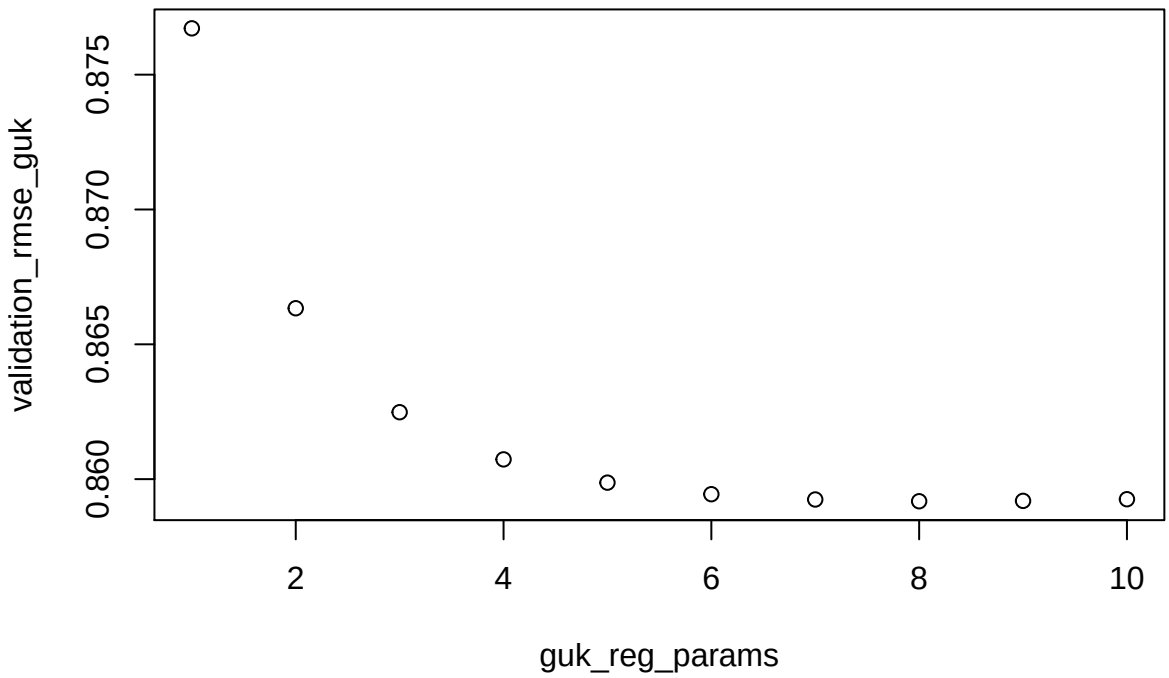



Then we try the same algorithm for a user regularization parameter of 5 and aim to find the optimal user|genre regularization parameter to overcome the regularization challenge we saw earlier, plot at the end, shows the optimum value to minimise the RMSE for the test set is to use a guk regularization parameter of 8, but given the difference with the other reg parameters is so low, we will go with the lower value (5), we do not want to apply too strong of a penalty in the genre score.

```
# Loop guk params
guk_reg_params = c(1:10)
validation_rmse_guk<-list()

for (guk_reg_param in guk_reg_params) {

  ### GUK Reg Parameter
  data_model<-edx_train
  val_data_model<-edx_test

  base_rating <- mean(data_model$rating)

  data_model <- data_model %>% mutate(
    br=base_rating,
    fcast_rating=br,
    resid=rating-fcast_rating
  )

  val_data_model <- val_data_model %>% mutate(
    br=base_rating,
```

```
  fcast_rating=br,
  resid=rating-fcast_rating
)

# This will remain constant
movies_coefs <-  data_model %>% as_tibble() %>% group_by(movieId) %>%
  summarise(mf = mean(resid)) %>%
  select(movieId, mf)

data_model<-merge(data_model, movies_coefs,
                  by.x = c("movieId"),
                  by.y = c("movieId"),
                  all.x = TRUE,
                  all.y = FALSE) %>%
  mutate(fcast_rating=br+mf, resid=rating-fcast_rating)


val_data_model<-val_data_model%>%mutate(br=base_rating)
val_data_model<-merge(val_data_model, movies_coefs,
                      by.x = c("movieId"),
                      by.y = c("movieId"),
                      all.x = TRUE,
                      all.y = FALSE)

val_data_model$mf[is.na(val_data_model$mf)]<-0
val_data_model<-val_data_model%>%mutate(fcast_rating=br+mf, resid=rating-fcast_rating)

user_reg_param <- 5
user_coefs <-  data_model %>% group_by(userId) %>%
  summarise(suf = sum(resid), n=n(), uf = suf/(user_reg_param + n)) %>%
  select(userId, uf)

data_model<-merge(data_model, user_coefs,
                  by.x = c("userId"),
                  by.y = c("userId"),
                  all.x = TRUE,
                  all.y = FALSE) %>%
  mutate(fcast_rating=br+mf+uf, resid=rating-fcast_rating)


val_data_model<-merge(val_data_model, user_coefs,
                      by.x = c("userId"),
                      by.y = c("userId"),
                      all.x = TRUE,
                      all.y = FALSE)

val_data_model$uf[is.na(val_data_model$uf)]<-0
val_data_model<-val_data_model%>%mutate(fcast_rating=br+mf+uf, resid=rating-fcast_rating)


guk_coefs <-  data_model %>% group_by(guk) %>%
  summarise(sgf = sum(resid), n=n(), gf = sgf/(guk_reg_param + n)) %>%
  select(guk, gf)
```

```
  data_model<-merge(data_model, guk_coefs,
                    by.x = c("guk"),
                    by.y = c("guk"),
                    all.x = TRUE,
                    all.y = FALSE) %>%
    mutate(fcast_rating=br+mf+uf+gf, resid=rating-fcast_rating)

  val_data_model<-merge(val_data_model, guk_coefs,
                        by.x = c("guk"),
                        by.y = c("guk"),
                        all.x = TRUE,
                        all.y = FALSE)

  # Fill Missing Values
  val_data_model$gf[is.na(val_data_model$gf)]<-0
  val_data_model<-val_data_model %>% mutate(fcast_rating=br+mf+uf+gf, resid=rating-fcast_rating)


  # RMSE(data_model$rating, data_model$fcast_rating)
  # RMSE(val_data_model$rating, val_data_model$fcast_rating)
  validation_rmse_guk[[guk_reg_param]]<-RMSE(val_data_model$rating, val_data_model$fcast_rating)

}


plot(guk_reg_params,validation_rmse_guk)
```




# Results

### Final model

Thus our final model to predict user score, is a combination of average score across all films/users, the average score each film is expected to have, the average score each user will give (regularization parameter of 5) and the average score each user will give to each genre type (regulatization parameter of 5), therefore if we built the model this way and calculated the RMSE on the validation set, this is what we would obtain,

```
# Final Model
data_model<-edx
val_data_model<-validation

base_rating <- mean(data_model$rating)

data_model <- data_model %>% mutate(
  br=base_rating,
  fcast_rating=br,
  resid=rating-fcast_rating
)

val_data_model <- val_data_model %>% mutate(
  br=base_rating,
  fcast_rating=br,
  resid=rating-fcast_rating
```

```
)

# This will remain constant
movies_coefs <-  data_model %>% as_tibble() %>% group_by(movieId) %>%
  summarise(mf = mean(resid)) %>%
  select(movieId, mf)

data_model<-merge(data_model, movies_coefs,
                  by.x = c("movieId"),
                  by.y = c("movieId"),
                  all.x = TRUE,
                  all.y = FALSE) %>%
  mutate(fcast_rating=br+mf, resid=rating-fcast_rating)


val_data_model<-val_data_model%>%mutate(br=base_rating)
val_data_model<-merge(val_data_model, movies_coefs,
                      by.x = c("movieId"),
                      by.y = c("movieId"),
                      all.x = TRUE,
                      all.y = FALSE)

val_data_model$mf[is.na(val_data_model$mf)]<-0
val_data_model<-val_data_model%>%mutate(fcast_rating=br+mf, resid=rating-fcast_rating)

user_reg_param <- 5
user_coefs <-  data_model %>% group_by(userId) %>%
  summarise(suf = sum(resid), n=n(), uf = suf/(user_reg_param + n)) %>%
  select(userId, uf)

data_model<-merge(data_model, user_coefs,
                  by.x = c("userId"),
                  by.y = c("userId"),
                  all.x = TRUE,
                  all.y = FALSE) %>%
  mutate(fcast_rating=br+mf+uf, resid=rating-fcast_rating)


val_data_model<-merge(val_data_model, user_coefs,
                      by.x = c("userId"),
                      by.y = c("userId"),
                      all.x = TRUE,
                      all.y = FALSE)

val_data_model$uf[is.na(val_data_model$uf)]<-0
val_data_model<-val_data_model%>%mutate(fcast_rating=br+mf+uf, resid=rating-fcast_rating)


guk_reg_param <-5
guk_coefs <-  data_model %>% group_by(guk) %>%
  summarise(sgf = sum(resid), n=n(), gf = sgf/(guk_reg_param + n)) %>%
  select(guk, gf)
```

```
data_model<-merge(data_model, guk_coefs,
                  by.x = c("guk"),
                  by.y = c("guk"),
                  all.x = TRUE,
                  all.y = FALSE) %>%
  mutate(fcast_rating=br+mf+uf+gf, resid=rating-fcast_rating)

val_data_model<-merge(val_data_model, guk_coefs,
                      by.x = c("guk"),
                      by.y = c("guk"),
                      all.x = TRUE,
                      all.y = FALSE)

# Fill Missing Values
val_data_model$gf[is.na(val_data_model$gf)]<-0
val_data_model<-val_data_model %>% mutate(fcast_rating=br+mf+uf+gf, resid=rating-fcast_rating)


RMSE(data_model$rating, data_model$fcast_rating)
```

```
## [1] 0.7536922
```

```
RMSE(val_data_model$rating, val_data_model$fcast_rating)
```

```
## [1] 0.8561964
```

The RMSE of the final model is:

- EDX dataset: 0.7536921573997194
- Validation: 0.8561963635140287

# Conclusion

On one hand, the size of the dataset and hardware limitations of home computers made impossible for a more accurate mathematical approach to be considered, as described in the executive summary the most basic form of one hot encoding greatly exceeded the computing capabilities of a regular desktop machine.

The RMSE at the end is till quite high (0.8562) as the residual standard deviation means on average if we assume the errors follow a normal distribution, the value any user will rate any will film could range between +/- 1.7124 our predicted score, which is a very wide error range (coeficient of variation = 0.24).

With more memory capabilities, and faster computing power we could look at ways of conducting:

- Dimensionality reduction, via film/user/genre clustering and/or PCA
- Model selection, using a model which models user scoring much closer to reality
- Model tuning, once selected our top performer model, we will need to fine tune it

On the other, the simplicity of the model allowed for a great interpretability, which usually makes sharing results and explaning results to others a lot easier, being in control of the model coefficients (instead of relying on pure mathematical optimization) also permits to increase the model scalability significantly (once

we move towards a *purer* mathematical solution it is usually very complex to re-adapt it, unless a lot of foresight is put into place, and still it is a very challenging problem).

To conclude, all things considered, the developed model feels a very interesting way of approaching expected user score, what it would be very interesting to record, and or measure, is the level of agreement the user have with our predicted scores, this is, when they see the forecast score, how compelled do they feel to watch the recommended movie, given sometimes this is not just about the right film score, but what is called UX (user experience design), and the forecast ratings play an important role in this whole experience, if the user feels the recommendation system provides good recommendations, regardless of our RMSE, the algorithm could be regarded as a good tool for its usage.