Capstone project on movielens from edX by VB

Vladimir Bousrez

2023-09-25

Capstone MovieLens

Part 1 - Introduction

In this project, we create a movie recommendation system leveraging on MovieLens. We will apply a machine learning algorithm to the training set of MovieLens and test it on a final hold-out set. The final hold-out test should not be used for training or selection of the algorithm, it should only be used at the end of the project on the final model. Instead, the 90% of data available for training from MovieLens, called edX, should be split further into train and test. The criteria to assess the success of the algorithm is the root-mean-square error (RMSE). RMSE measures the predictive power of a model by comparing the gaps between the predictions and the actual values. It is the square root of the average of squared errors. In order to proceed, we will:

- Prepare the work environment:
- Download and load the libraries.
- Download, unzip and consolidate the movie and ratings files.
- Dedicate 10% of the "movielens" file to final testing in the final_holdout_test. The remaining 90% goes to "edX".
- Split "edX" between test and train.

Analyse the data set.

- Calculate the RMSE for four models.
- Simple ratings average
- Movie effects
- User effects
- Genre effects.
- Including the regularization and a cross validation.
- Calculation of RMSE on the final model.

Part 2 – Methods/analysis.

Downloading and loading the files

The steps to download and load the libraries are provided by the course. The movielens data set will be split between a edX set and a 10% validation set "final holdout set", which will be used to calculate the RMSE of the final model selected.

Create edx and final_holdout_test sets

```
#Note: this process could take a couple of minutes
# Installing the packages
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
```

```
## Loading required package: tidyverse

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

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

```
## Loading required package: caret
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
# library
library(tidyverse)
library(caret)
library(dslabs)

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

# Download the files
options(timeout = 600)
dl <- "ml-10M100K.zip"
```

```r
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

# Unzip rating file
ratings_file <- "ml-10M100K/ratings.dat"
if(!file.exists(ratings_file))
  unzip(dl, ratings_file)
file.exists(dl)
```

```
## [1] TRUE
```

```r
#Unzip movie file
movies_file <- "ml-10M100K/movies.dat"
if(!file.exists(movies_file))
  unzip(dl, movies_file)

#Load and convert rating matrix into data frame
ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)

colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))


#Load and convert movie matrix into data frame
movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))

#Join tables ratings and movies to build movie lens
movielens <- left_join(ratings, movies, by = "movieId")
```

Review of movielens

The outcome from the code above is the "movielens" data set. It has: - 10,000,054 rows and 6 columns - 69,878 unique user id. - 10,677 unique movie id.

Making processing faster

We add a step which is not in the edX instruction for faster processing and convert string character of genre into string of integer for faster processing

```r
movielens$genres = as.factor(movielens$genres)
movielens$genres = as.numeric(movielens$genres)
movielens$genres = as.character(movielens$genres)
```

Create a final hold_out_set

In order to validate the final model with 10% of the initial data, we create the "final_holdout_test". We will need to make sure that the final_holdout_test does not include data which are not in the data set, "edx", for training and testing, in order to not biais the results of the model.

```
# Final hold-out test set will be 10% of MovieLens data
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
```

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

```
# set.seed(1) # if using R 3.5 or earlier
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 final hold-out test set are also in edx set
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")

# Add rows removed from final hold-out test set back into edx set
removed <- anti_join(temp, final_holdout_test)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

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


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

#This is what the edX table looks like:
head(edx)
```

```
##   userId movieId rating timestamp                         title genres
## 1      1     122      5 838985046              Boomerang (1992)    577
## 2      1     185      5 838983525               Net, The (1995)    187
## 4      1     292      5 838983421               Outbreak (1995)    210
## 5      1     316      5 838983392               Stargate (1994)     98
## 6      1     329      5 838983392 Star Trek: Generations (1994)     71
## 7      1     355      5 838984474       Flintstones, The (1994)    460
```

Split between edX train and test

In order to avoid using final_holdout_test for testing the different models, we will spit edX between test and train. Edx_train will be used to train the model while edx_test will be used to test the different models and select which model works best. We will also make sure that no user or movie included only in edx_test and not in edx_train could disturb the results.

```r
## Split edX between test and train for models 1 to 4

test_index <- createDataPartition(y = edx$rating, times = 1, p = 0.1, list = FALSE)
edx_train <- edx[-test_index,]
temp <- edx[test_index,]
# Make sure userId and movieId in test set are also in train set
edx_test  <- temp%>%
  semi_join(edx_train, by = "movieId") %>%
  semi_join(edx_train, by = "userId")

# Add rows removed from test set back into train set
removed <- anti_join(temp, edx_test)
```

```
## Joining with `by = join_by(userId, movieId, rating, timestamp, title, genres)`
```

```r
edx_train <- rbind(edx_train, removed )
rm(temp, removed)
dim(edx_train)
```

```
## [1] 8100067       6
```

```r
dim(edx_test)
```

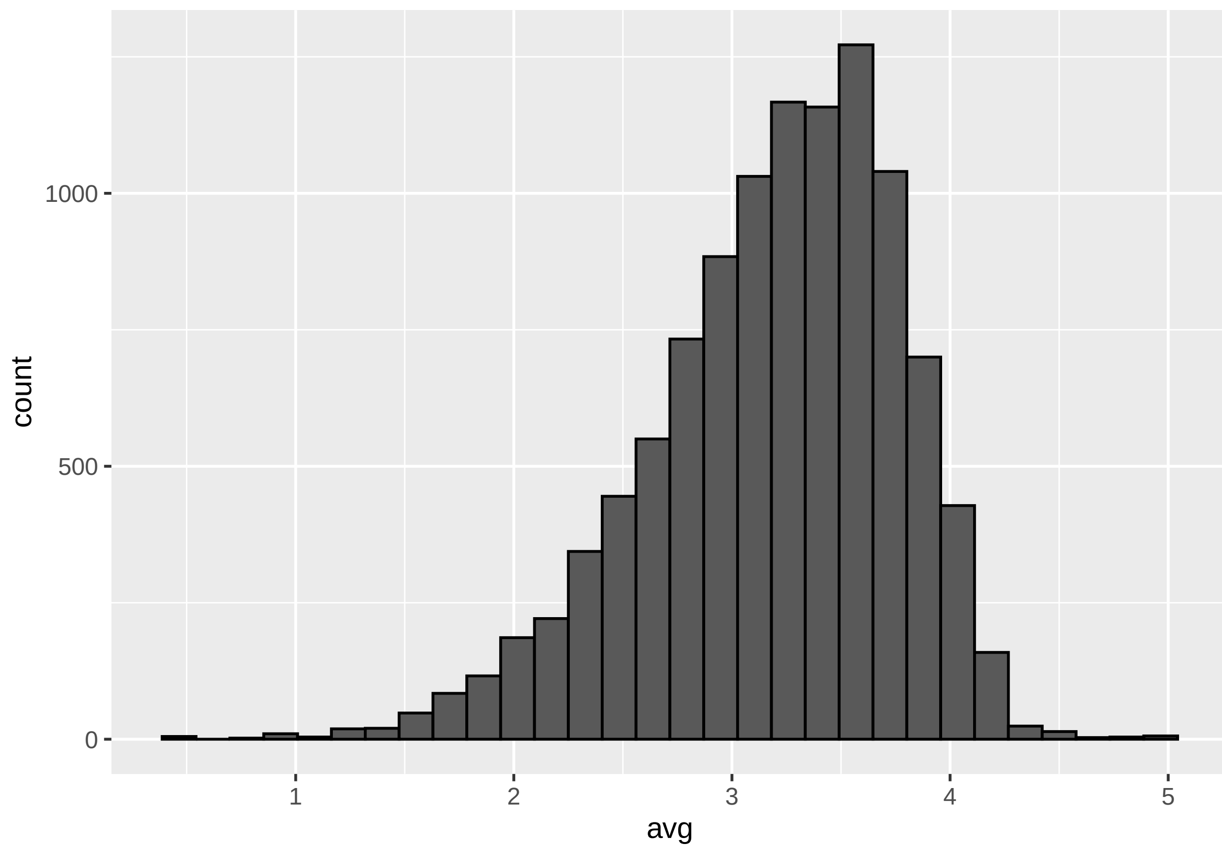
```
## [1] 899988      6
```

Analyze the dataset

Before proceeding with calculating the models, we want to assess which variable makes more sense to include. We will look at movie, users, timestamp and genre. We will look at how widespread is the rating for each variable and what is the distribution of ratings for each variable. We will thus look at the distribution of ratings for each variable. - Movie effect

```r
#plotting the average rating for those above 100 ratings to assess distribution and concentration of ra
```

```r
edx%>%
  group_by(movieId)%>%
  summarize(avg= mean(rating))%>%
  filter(n()>=100)%>%
  ggplot(aes(avg))+
  geom_histogram(bins=30, color = "black")
```



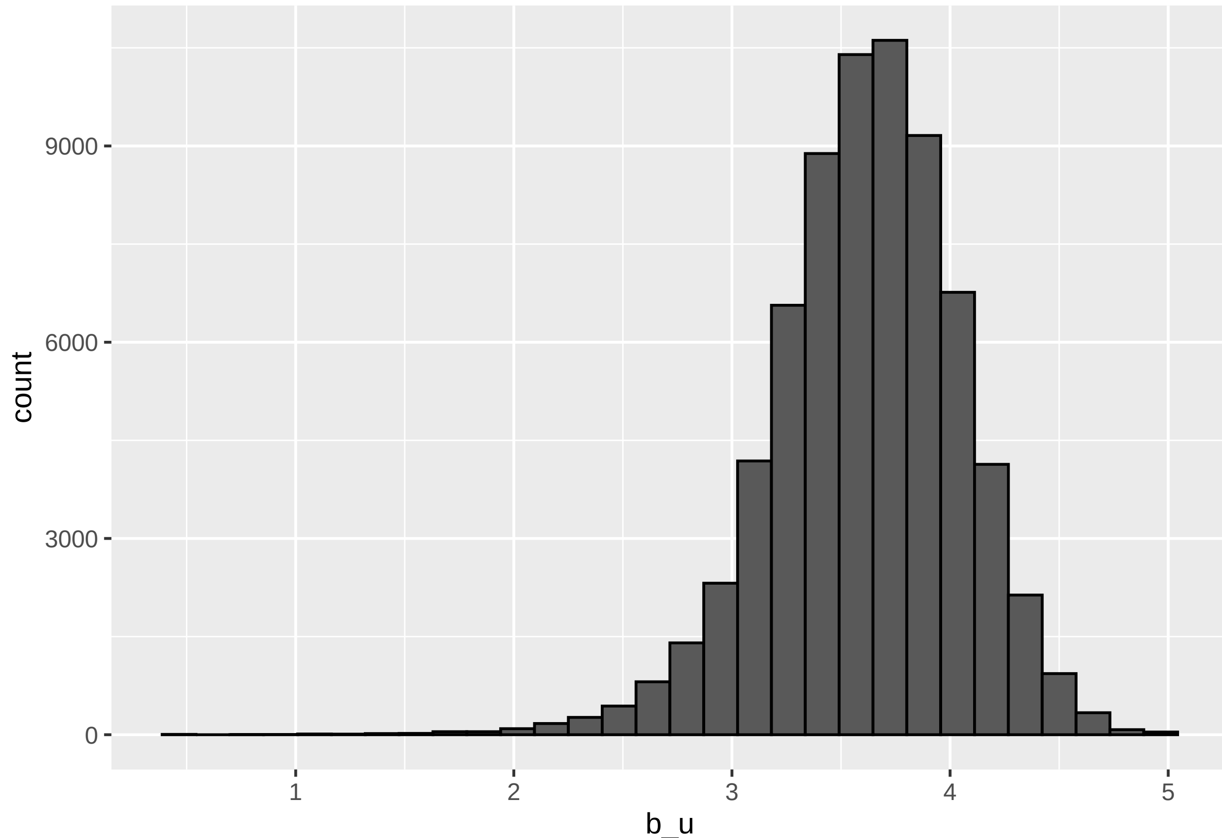
We can see a great variety of ratings in the movie ratings value and that this will surely be a significant indicator we will test.

- User effect

```
#Analysis of user effect
names(edx)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

```
#plotting the average rating for those above 100 ratings to assess distribution and concentration of ra
edx%>%
  group_by(userId)%>%
  summarize(b_u= mean(rating))%>%
  filter(n()>=100)%>%
  ggplot(aes(b_u))+
  geom_histogram(bins=30, color = "black")
```



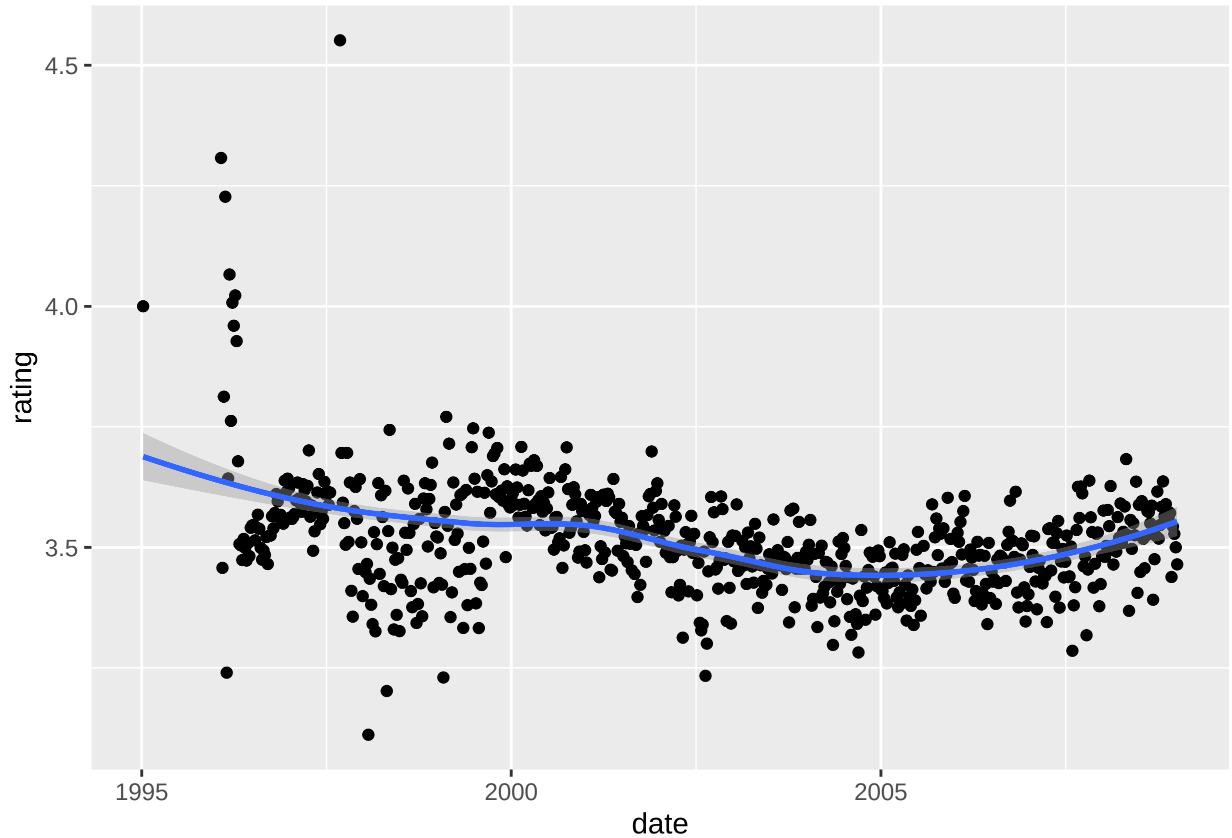
We can see a great variety of ratings in the user ratings value and that this will surely be a significant indicator we will test.

- Time effect

```
#Assessing opportunity for using time as predictor
#adding timestamp
edx2 <- mutate(edx_train, date= as_datetime(timestamp))

edx2 %>%
  mutate(date= round_date(date, unit = "week")) %>%
  group_by(date)%>%
  summarize(rating=mean(rating))%>%
  ggplot(aes(date, rating))+
  geom_point()+
  geom_smooth()
```

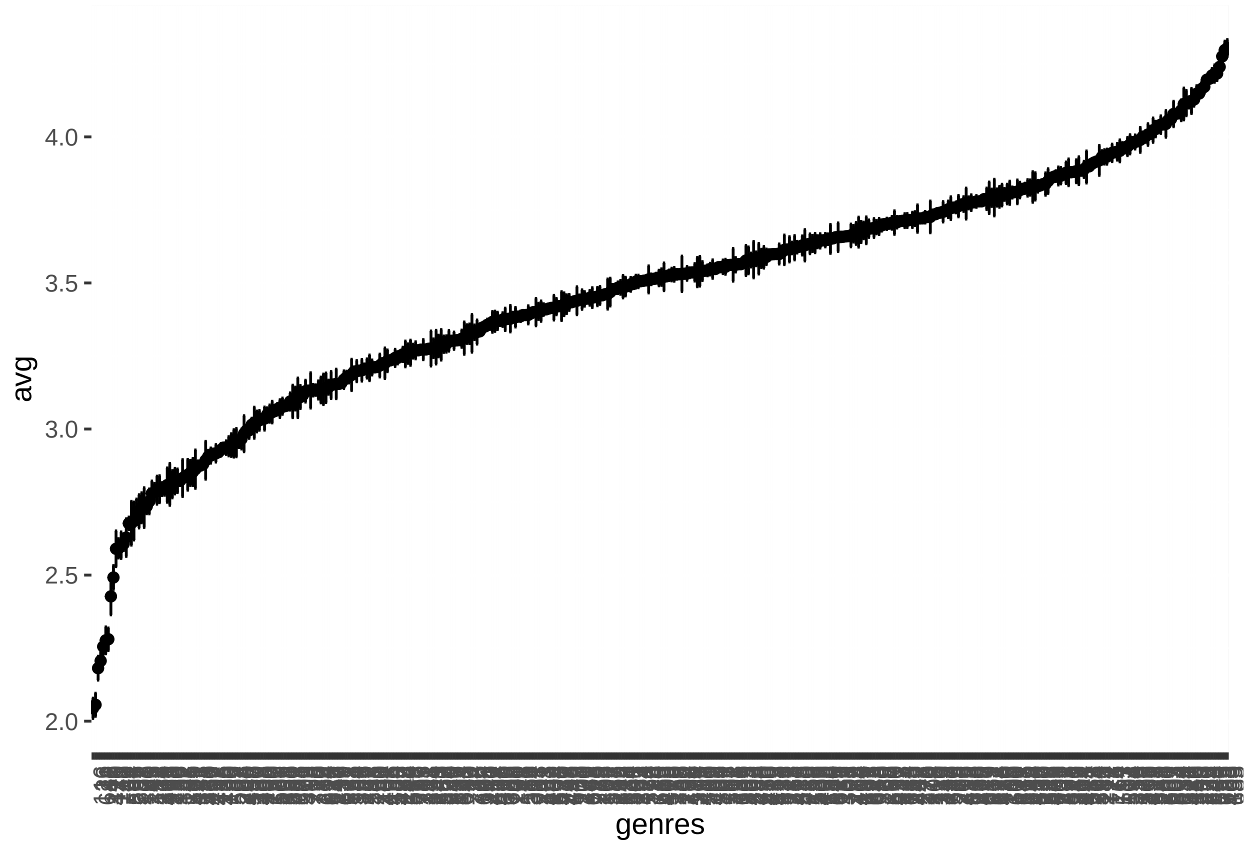
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



We can see the effect is not so strong, we will not keep it an indicator to test.

- Genre effect

```
edx %>%
  group_by(genres) %>%
  summarize(n=n(), avg= mean(rating), se = sd(rating)/sqrt(n())) %>%
  filter(n >=1000) %>%
  mutate(genres = reorder(genres, avg)) %>%
  ggplot(aes(x= genres, y=avg, ymin= avg - 2*se, ymax=avg+2*se))+
  geom_point()+
  geom_errorbar()+
  theme(axis.text.x = element_text(angle =90, hjust=1))
```



This will be kept as a rating to test.

##"Training" and "Testing" the models

Model 1 based on simple average rating

We will start with using the average of all movie ratings as a basis predictor.

```
  # Computing
muedx <- mean(edx_train$rating)
  #Assessing prediction power
model_1_rmse <- RMSE(edx_test$rating, muedx)
model_1_rmse
```

```
## [1] 1.061135
```

Any variable which will improve the accuracy from this RMSE will add value to the algorithm.

Model 2 with movie effect

We will see if using the average rating for each movie improves the RMSE. That means that we will calculate the average rating for each movie excluding any other variable. In order to proceed, we will not use the lm function which might take too long but rely on the directly on the known expression of the estimate as averages as if obtained by the least square, leveraging on "ratings – muedx".

```
#Compute least square to estimate using ratings-muedx
names(edx_train)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

```
movie_avgs <- edx_train%>%
  group_by(movieId)%>%
  summarize(b_i= mean(rating - muedx))
#Assessing improvement in prediction
predicted_ratings <- muedx +
  edx_test %>%
  left_join(movie_avgs, by= 'movieId') %>%
  pull(b_i)

model_2_rmse <- RMSE(predicted_ratings, edx_test$rating)
model_2_rmse
```

```
## [1] 0.9441568
```

The RMSE has improved.

Model 3 with user effect

Here, we will see if including the average rating per user can improve the RMSE. The lm function will also take too long so we rely on the directly on the known expression of the estimate as averages as if obtained by the least square leveraging on "ratings -u__hat-bi__hat".

```
#Compute approximation of least square using average ratings-u_hat-bi_hat

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

#construct predictors
predicted_ratings<-edx_test %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  mutate(pred= muedx + b_i+ b_u) %>%
  pull(pred)
#measures improvement
model_3_rmse <- RMSE(predicted_ratings,edx_test$rating)

model_3_rmse
```

```
## [1] 0.8659736
```

Using the average rating per user, in addition to the rating per movie and the overall average improve the MSME.

Model 4 with genre effect

Here, we will see if including the average rating per user can improve the RMSE. The lm function will also take too long so we will leverage on "ratings -u_hat-bi_hat-bu_hat".

```
#Compute rating per genre
names(edx_train)
```

```
## [1] "userId"    "movieId"   "rating"    "timestamp" "title"     "genres"
```

```
genre_avgs <- edx_train %>%
  left_join(movie_avgs, by='movieId') %>%
  left_join(user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(b_g = mean(rating - muedx - b_i - b_u))

  #Build prediction
predicted_ratings <- edx_test %>%
  left_join(movie_avgs, by= 'movieId') %>%
  left_join(user_avgs, by='userId') %>%
  left_join(genre_avgs, by='genres') %>%
  mutate(pred = muedx + b_i + b_u + b_g) %>%
  pull(pred)

model_4_rmse <- RMSE(predicted_ratings,edx_test$rating)
model_4_rmse
```

```
## [1] 0.8656018
```

Using the genre enables to improve the RMSE.

Model 5 with regularization and cross validation

Including the regularization and the cross validation. The number of users impacts the quality of the prediction, the fewer users give a rating, the lower is the quality of the prediction. We use regularization to reduce the impact of outliers created using reduced sample size. We will determine the lambda which enable to minimize the RMSE: - First, we set up the function to calculate RMSE based on lambda

```
# Compute the predicted ratings on validation dataset using different values of lambda

# Definition of function to compute RMSE for one lambda
do_one_rmse_lambda <- function(lambda, edx_train, edx_test)  {
  # Calculate the average by movie
  movie_avgs_lambda <- edx_train %>%
    group_by(movieId) %>%
    summarize(b_i = sum(rating - muedx) / (n() + lambda))

  # Calculate the average by user
  user_avgs_lamda <- edx_train %>%
    left_join(movie_avgs_lambda, by='movieId') %>%
    group_by(userId) %>%
```

```r
    summarize(b_u = sum(rating - b_i - muedx) / (n() + lambda))

  # Calculate the average by genre
  genre_avgs_lamda <- edx_train %>%
    left_join(movie_avgs_lambda, by='movieId') %>%
    left_join(user_avgs_lamda, by='userId') %>%
    group_by(genres) %>%
    summarize(b_u_g = sum(rating - b_i - muedx - b_u) / (n() + lambda))

  # filter rows where effects are in train but not in test
  fg = edx_test$genres %in% unique(edx_train$genres)
  fu = edx_test$userId %in% unique(edx_train$userId)
  fm = edx_test$movieId %in% unique(edx_train$movieId)
  edx_test = edx_test[fg & fu & fm,]

  # Compute the predicted ratings on test dataset
  predicted_ratings <- edx_test %>%
    left_join(movie_avgs_lambda, by='movieId') %>%
    left_join(user_avgs_lamda, by='userId') %>%
    left_join(genre_avgs_lamda, by='genres') %>%
    mutate(pred = muedx + b_i + b_u + b_u_g) %>%
    pull(pred)

  # Predict the RMSE on the validation set
  rmse = RMSE(edx_test$rating, predicted_ratings)

  rmse
}
```

- Then, in order to reduce the bias for the error of prediction, we cross validate the results by splitting edx into 10 train and test sets and assess which lambda minimizes the RMSE .

```r
# Implementation of cross-validation for setting best lambda, after finding lambda from only sample edx
# Then apply to the final validation sample
# (final_holdout_test at the end only)

lambdas <- seq(0, 8, 0.5)
lambdas
```

```
##  [1] 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5 8.0
```

```r
rmses   <- 0 * lambdas
rmses
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```r
# with cross validation
#first find the k folds from edx

idx = as.data.frame(1:nrow(edx))
colnames(idx) = NULL
#Split the sample in 10
```

```r
K = 3

set.seed(543210)
folds <- sample( cut(seq(1,nrow(edx)),breaks=K,labels=FALSE) )

table(folds)
```

```
## folds
##       1       2       3
## 3000019 3000018 3000018
```

```r
sum(table(folds))
```

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

```r
head(folds)
```

```
## [1] 2 3 1 3 1 2
```

```r
tail(folds)
```

```
## [1] 3 3 3 1 3 2
```

```r
#then apply compute rmse for each fold removed, and average
for (l in 1:length(lambdas)) {
  print(l)
  lambda = lambdas[l]
  remse_l = 0

  for (k in 1:K) {
    # print(k)
    cat(".")
    edx_train_k = edx[folds!=k,]
    edx_test_k = edx[folds==k,]
    # print(dim(edx_train_k))
    # print(dim(edx_test_k))
    remse_lk = do_one_rmse_lambda (lambda=lambda,
                                   edx_train=edx_train_k,
                                   edx_test=edx_test_k)

    remse_l = remse_l + remse_lk / K

  rm(edx_train_k)
    rm(edx_test_k)
  }

  rmses[l] = remse_l
  print(paste( lambda,  remse_l, sep= " "))
}
```

```
## [1] 1
## ...[1] "0 0.866992649238004"
## [1] 2
## ...[1] "0.5 0.866749714182458"
## [1] 3
## ...[1] "1 0.866568075812452"
## [1] 4
## ...[1] "1.5 0.86642709372186"
## [1] 5
## ...[1] "2 0.866317642635574"
## [1] 6
## ...[1] "2.5 0.866234097804227"
## [1] 7
## ...[1] "3 0.866172445357726"
## [1] 8
## ...[1] "3.5 0.86612960361804"
## [1] 9
## ...[1] "4 0.86610310680585"
## [1] 10
## ...[1] "4.5 0.866090929972203"
## [1] 11
## ...[1] "5 0.866091380187688"
## [1] 12
## ...[1] "5.5 0.866103023322718"
## [1] 13
## ...[1] "6 0.866124631959345"
## [1] 14
## ...[1] "6.5 0.866155146822132"
## [1] 15
## ...[1] "7 0.866193647357907"
## [1] 16
## ...[1] "7.5 0.866239328778008"
## [1] 17
## ...[1] "8 0.866291483819529"
```

```r
rmses_withcv = rmses

# Get the lambda value that minimize the RMSE
min_lambda <- lambdas[which.min(rmses)]
min_lambda
```

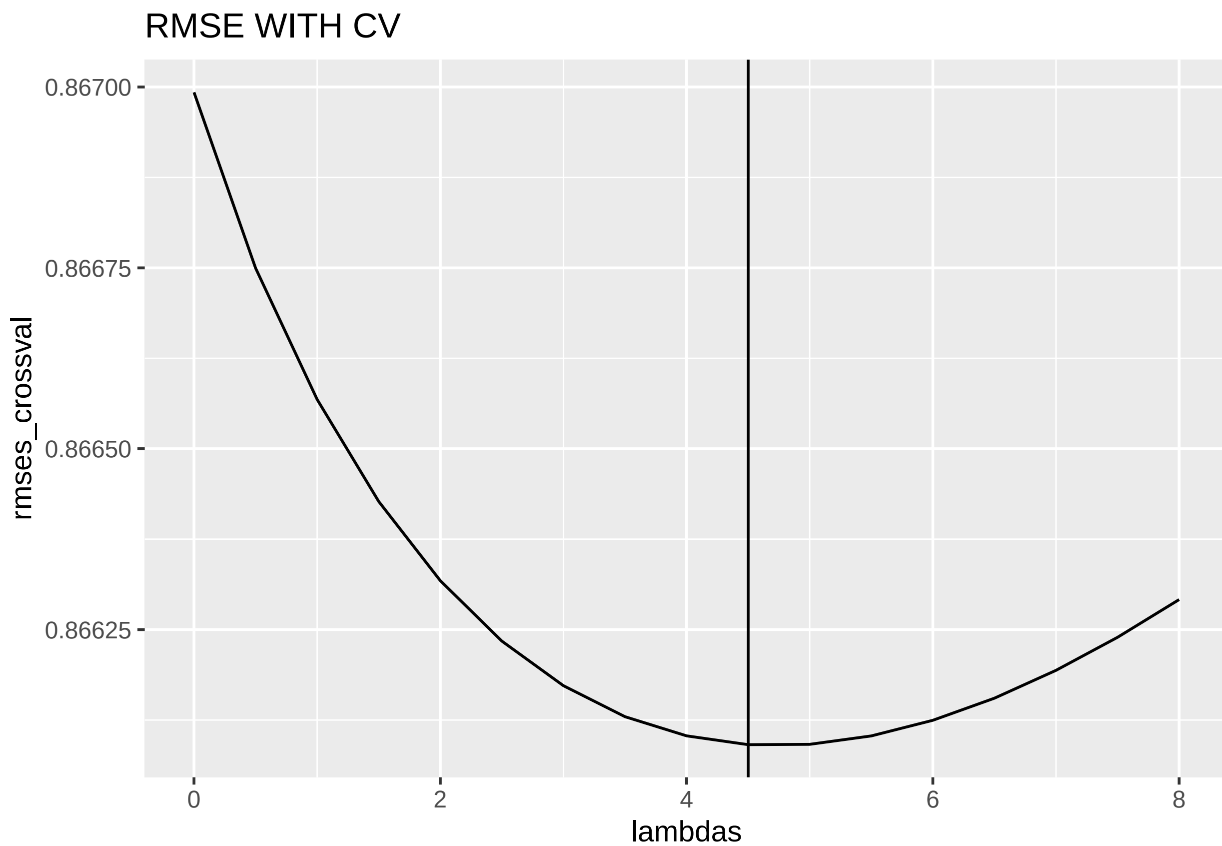
```
## [1] 4.5
```

```r
min_lambda
```

```
## [1] 4.5
```

```r
library(ggplot2)
datap <- data.frame(lambdas=lambdas,
                    rmses_crossval=rmses_withcv)
ggplot(datap, aes(lambdas, rmses_crossval)) +
  ggtitle("RMSE WITH CV")+
```

```r
  geom_line()+
  geom_vline(xintercept = min_lambda)
```




```r
#Compute rmse on edx train and test sample

model_5_rmse =
  remse_lk = do_one_rmse_lambda (lambda=min_lambda,
                                 edx_train=edx_train,
                                 edx_test=edx_test)
```

Part 3 – Results.

We obtain the following results: The lambda which minimizes RMSE is equal to 5.

```r
min_lambda
```

```
## [1] 4.5
```

We can see that the best RMSE effectively comes from model 5:

```r
model_5_rmse
```

```
## [1] 0.8651295
```

```
model_4_rmse
```

```
## [1] 0.8656018
```

```
model_3_rmse
```

```
## [1] 0.8659736
```

```
model_2_rmse
```

```
## [1] 0.9441568
```

```
model_1_rmse
```

```
## [1] 1.061135
```

Final verification of Model 5 RMSE on validation set

```
model_final_rmse =
  remse_lk = do_one_rmse_lambda (lambda=min_lambda,
                                 edx_train=edx,
                                 edx_test=final_holdout_test)
model_final_rmse
```

```
## [1] 0.8644543
```

On checking on final validation test, we see that the RMSE for the selected model 5 is equal to 0.8644. > model_final_rmse [1] 0.8644501

Using average, movie effect, user effect and genre effect combined with regularization has proven the best method with a RMSE on the validation set of 0.8644501.

Part 4 – Conclusion.

We have built a model that predicts the ratings for movielens with an accuracy, measured by RMSE, within the expectations of the assignment (<0.86490). This has been achieved by including three variables: user, movie and genre, as well as including the regularization for the ratings given by a few numbers of users. Further improvement could be achieved by splitting further the genre variable, which is currently very aggregated and including the time effect (although this should have marginal effect), as well as using a r function relevant for big data on the contrary to the old function lm which is relevant only for small dataset.