1st_Harvard_project

Konstantinos Roinas

16/6/2021

INTRODUCTION

In this project we will work with movielens dataset (10M lines). We will break it down in 2 parts edx and validation. The first will be used for testing training purposes and the validation part only at the end to calculate the final criterion of success or not of our model. The criterion will be a low RMSE. We will build our model as continuation of the one showed within the lecture, adding and testing more factors(predictors). Always trying to predict the rating that a specific user gives to a specific movie.

LOADING OUR DATASET

Installing libraries that will be used.

```
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   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## Warning: package 'stringr' was built under R version 4.0.5
```

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

```
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
```

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

```
## 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
```

```
library(tidyverse)
library(caret)
library(data.table)
```

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

```
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)
```

ANALYSIS
Lets see edx contents and dimensions

```
dim(edx)
```

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

```
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
```

It has more than 9 million samples and columns like title, timestamp (of rating), userId (the one’s that rates), movieId and title (of rated movie), genres (category) and of course rating
We have to create test and train set from edx. Here against Pareto principle we will use a ratio 90/10 train / set in order to have a test set comparable to validation one.

```
test_index <- createDataPartition(edx$rating, times = 1, p = 0.1, list = FALSE)
test_set <- edx%>% slice(test_index)
train_set<- edx%>% slice(-test_index)
```

CREATION OF NEW COLLUMNS (VARIABLES - POSSIBLE FACTORS) We decided create new variables and test our model with them. Of course we will apply changes in both sets (train and set) NRPM (number of ratings per movie), NRPU (number of ratings per user), MA (movie age - difference between year of rating and production year), MR (month of rating), HR (hour of rating).

```
train_set<-train_set%>% mutate(Rating_date=as.Date(as.POSIXct(timestamp, origin = "1970-01-0
1") ))
train_set<-train_set%>% mutate(RY=as.numeric(format(Rating_date, '%Y')))
test_set<-test_set%>% mutate(Rating_date=as.Date(as.POSIXct(timestamp, origin = "1970-01-01")
))
test_set<-test_set%>% mutate(RY=as.numeric(format(Rating_date, '%Y')))
```

To find movie age (MA) we have to reduce Rating year (RY) by movie year(MY)

```
train_set<-train_set%>%mutate(MY=substr(title,nchar(title)-4,nchar(title)-1))
train_set<- train_set%>%mutate(MA=RY-as.numeric(MY))
test_set<-test_set%>%mutate(MY=substr(title,nchar(title)-4,nchar(title)-1))
test_set<- test_set%>%mutate(MA=RY-as.numeric(MY))
```

To light our dataset at this point we throw away the following columns that we will not use or no need them anymore: title, MY, RY.

```
test_set$title<-NULL
test_set$MY<-NULL
test_set$RY<-NULL
train_set$title<-NULL
train_set$MY<-NULL
train_set$RY<-NULL
test_set<-test_set%>% mutate(MR=as.numeric(format(Rating_date, '%m')))
test_set<-test_set%>% group_by(userId) %>% mutate(NRPU=n())
test_set<-test_set%>% group_by(movieId)%>% mutate(NRPM=n())
train_set<-train_set%>% mutate(MR=as.numeric(format(Rating_date, '%m')))
train_set<-train_set%>% group_by(userId) %>% mutate(NRPU=n())
train_set<-train_set%>% group_by(movieId)%>% mutate(NRPM=n())
```

So we have all our variables except the Hour of Rating HR, that we will extract from timestamp. Lets throw away Rating_date that is no needed now.

```
train_set$Rating_date<-NULL
train_set<-train_set%>% mutate(Rating_date_time=as.POSIXct(timestamp, origin="1970-01-01"))
test_set$Rating_date<-NULL
test_set<-test_set%>% mutate(Rating_date_time=as.POSIXct(timestamp, origin="1970-01-01"))
```

So we have extract from timestamp(UNIX mode), Rating_date_time.

```
train_set<-train_set%>% mutate(HR=as.numeric(format(Rating_date_time, '%H')))
test_set<-test_set%>% mutate(HR=as.numeric(format(Rating_date_time, '%H')))
```

So now we have all our variables and we can delete timestamp and Rating_date_time.

```
train_set$timestamp<-NULL
train_set$Rating_date_time<-NULL
test_set$timestamp<-NULL
test_set$Rating_date_time<-NULL
```

INVESTIGATING IF NEW VARIABLES AFFECT RATING

No need to use the whole edx or train_set 8-9 million lines for this check. Lets use a sample of it.

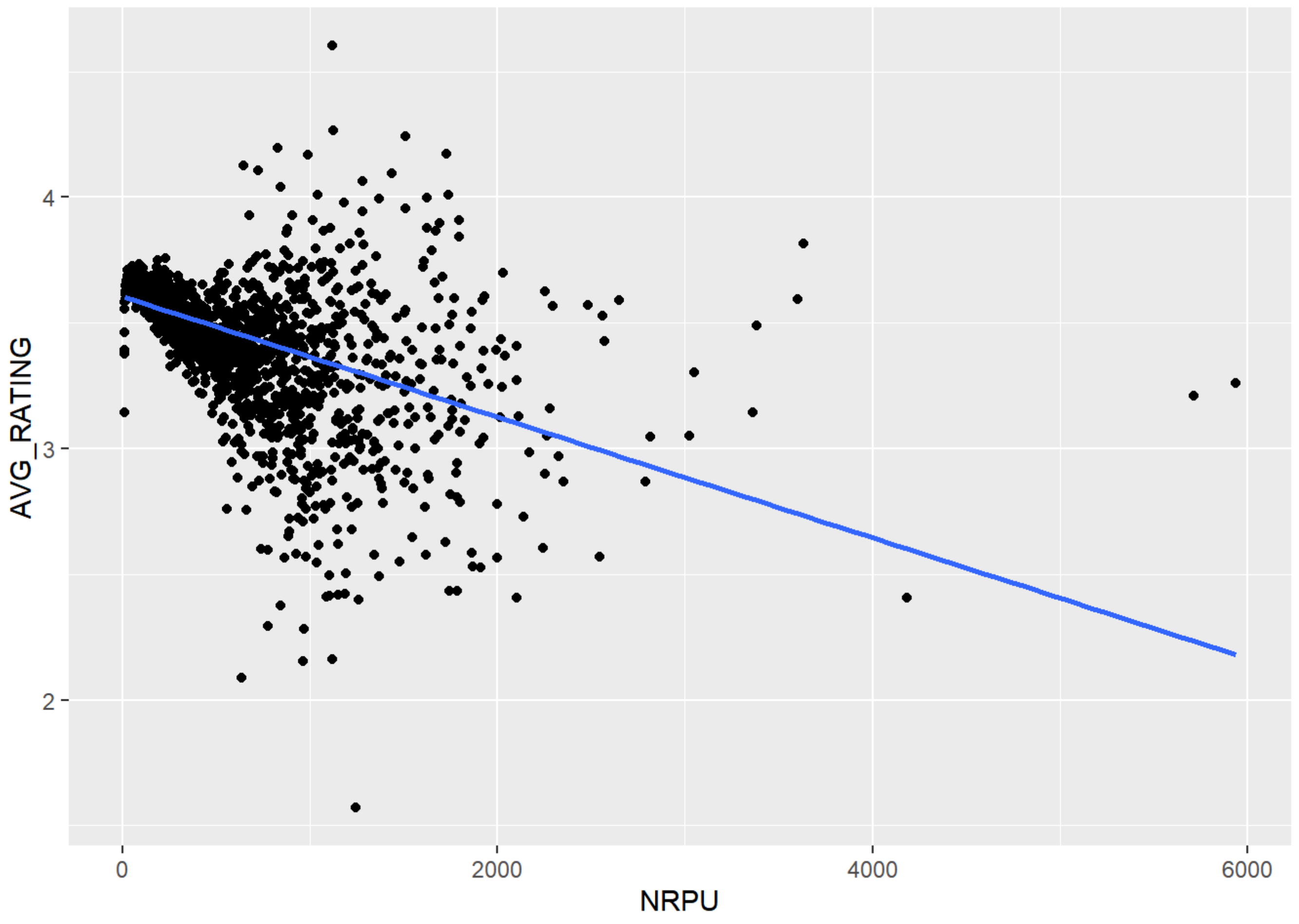
```
train_set_sample<-train_set %>% sample_frac(0.1)
dim(train_set_sample)
```

```
## [1] 809877      9
```

```
train_set_sample%>%group_by(NRPU)%>%summarize(NRPU,AVG_RATING=mean(rating))%>%ggplot(aes(NRP
U,AVG_RATING))+geom_point()+geom_smooth(method = "lm")
```

```
## `summarise()` regrouping output by 'NRPU' (override with `.groups` argument)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

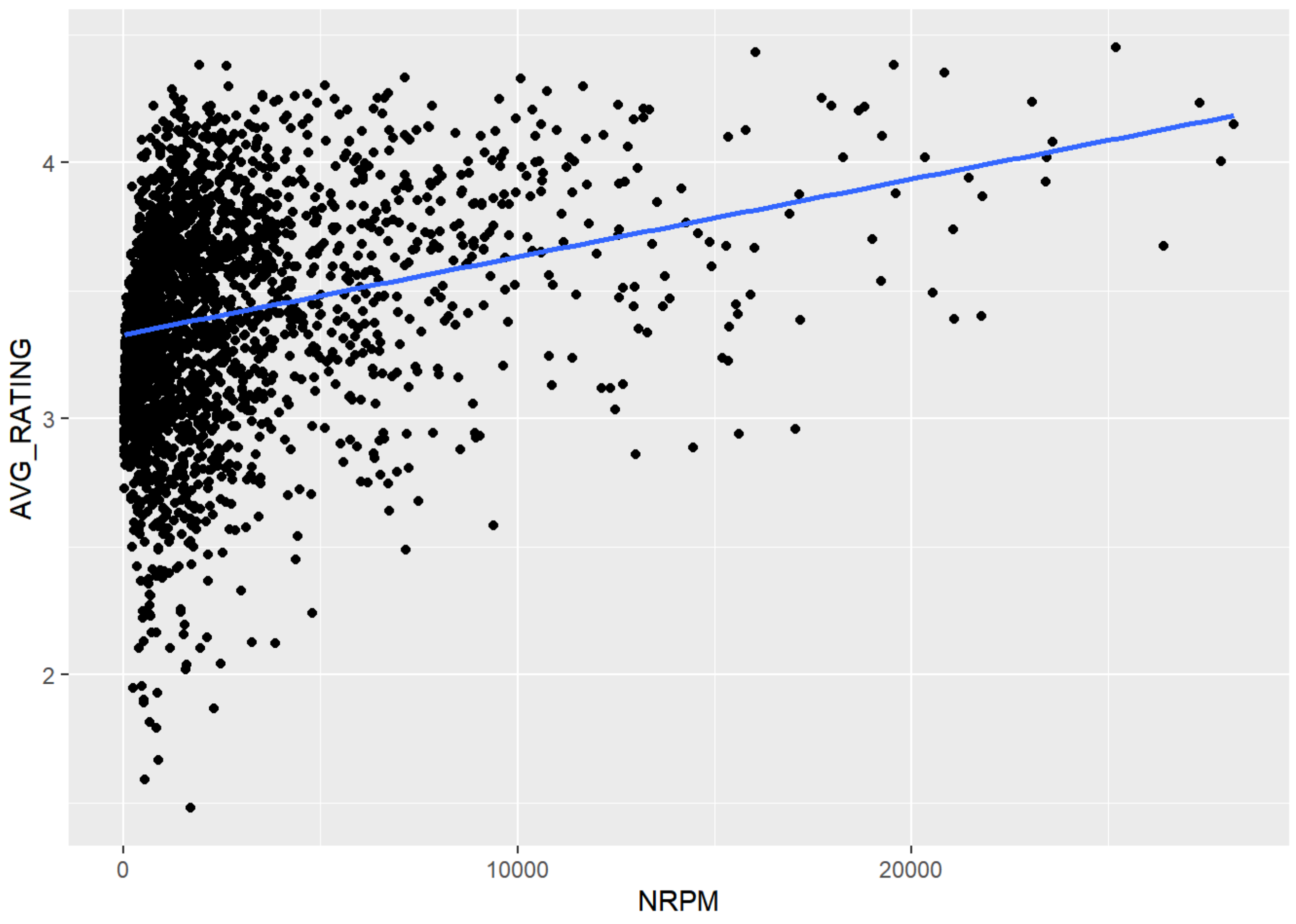


We see the more a user do rating the stricter is, that sounds normal. From few rating and average more than 3.5 we reach users with 3000 ratings and a bit less than 3 average.

```
train_set_sample%>%group_by(NRPM)%>%summarize(NRPM,AVG_RATING=mean(rating))%>%ggplot(aes(NRP
M,AVG_RATING))+geom_point()+geom_smooth(method = "lm")
```

```
## `summarise()` regrouping output by 'NRPM' (override with `.groups` argument)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

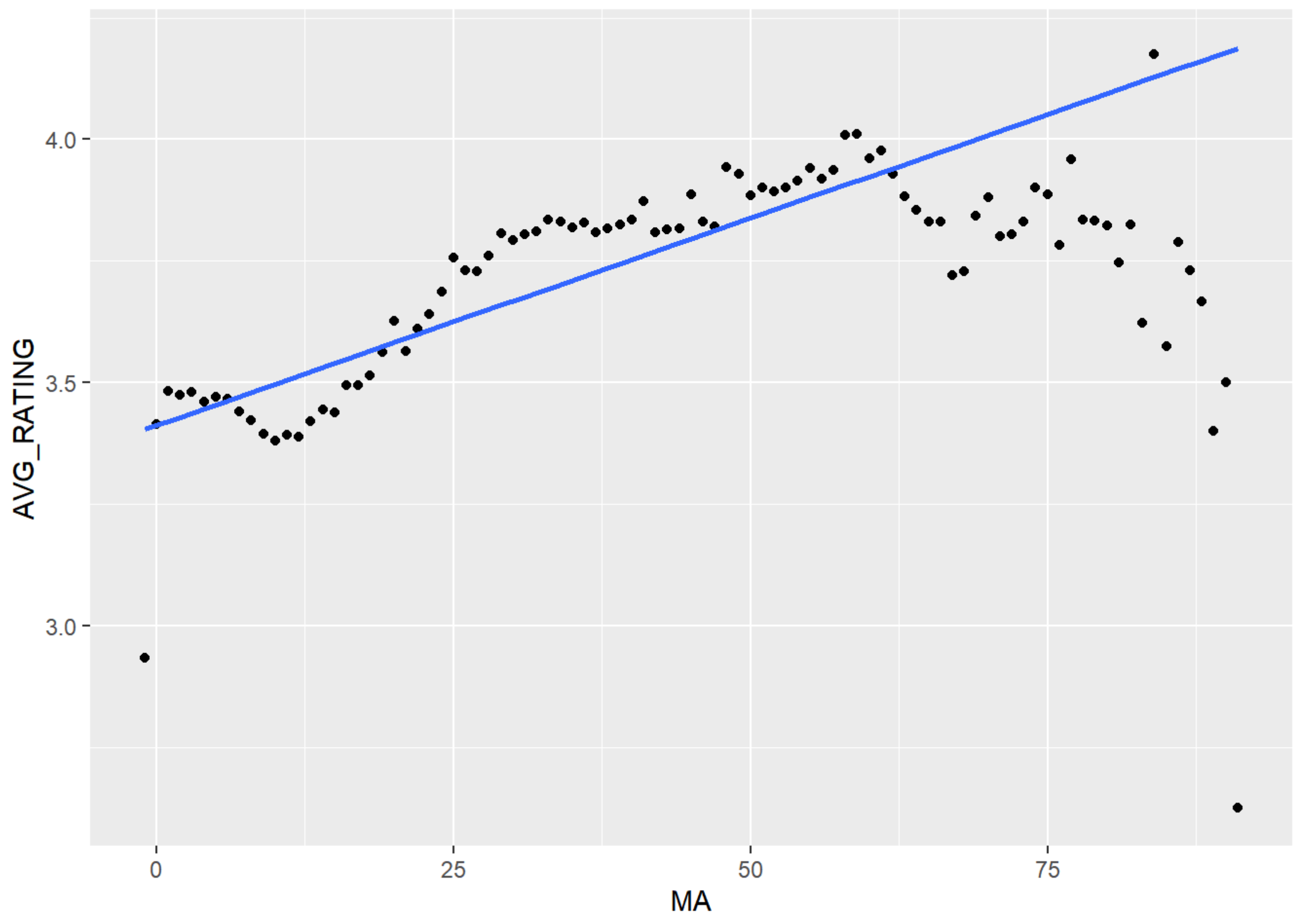


Here the more ratings has a movie the higher they are, that also sounds normal as masterpieces take more ratings in average and in number.

```
train_set_sample%>%group_by(MA)%>%summarize(MA,AVG_RATING=mean(rating))%>%ggplot(aes(MA,AVG_R
ATING))+geom_point()+geom_smooth(method = "lm")
```

```
## `summarise()` regrouping output by 'MA' (override with `.groups` argument)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

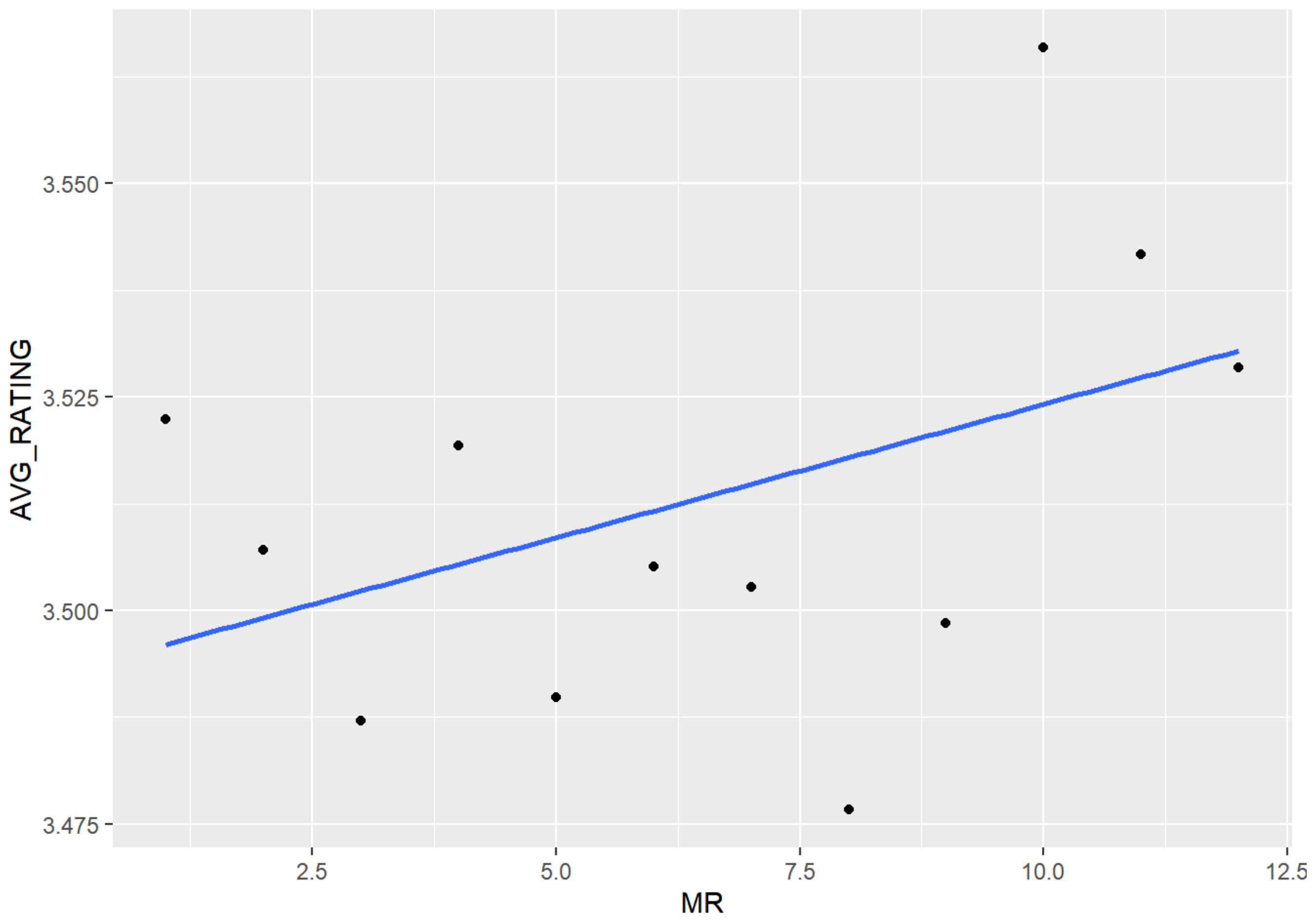


The older the movie the better ratings. Expected as older movies are considered of higher quality. So we start from 3.5 rating in new ones and reach 4 to the older ones.

```
train_set_sample%>%group_by(MR)%>%summarize(MR,AVG_RATING=mean(rating))%>%ggplot(aes(MR,AVG_R
ATING))+geom_point()+geom_smooth(method = "lm")
```

```
## `summarise()` regrouping output by 'MR' (override with `.groups` argument)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

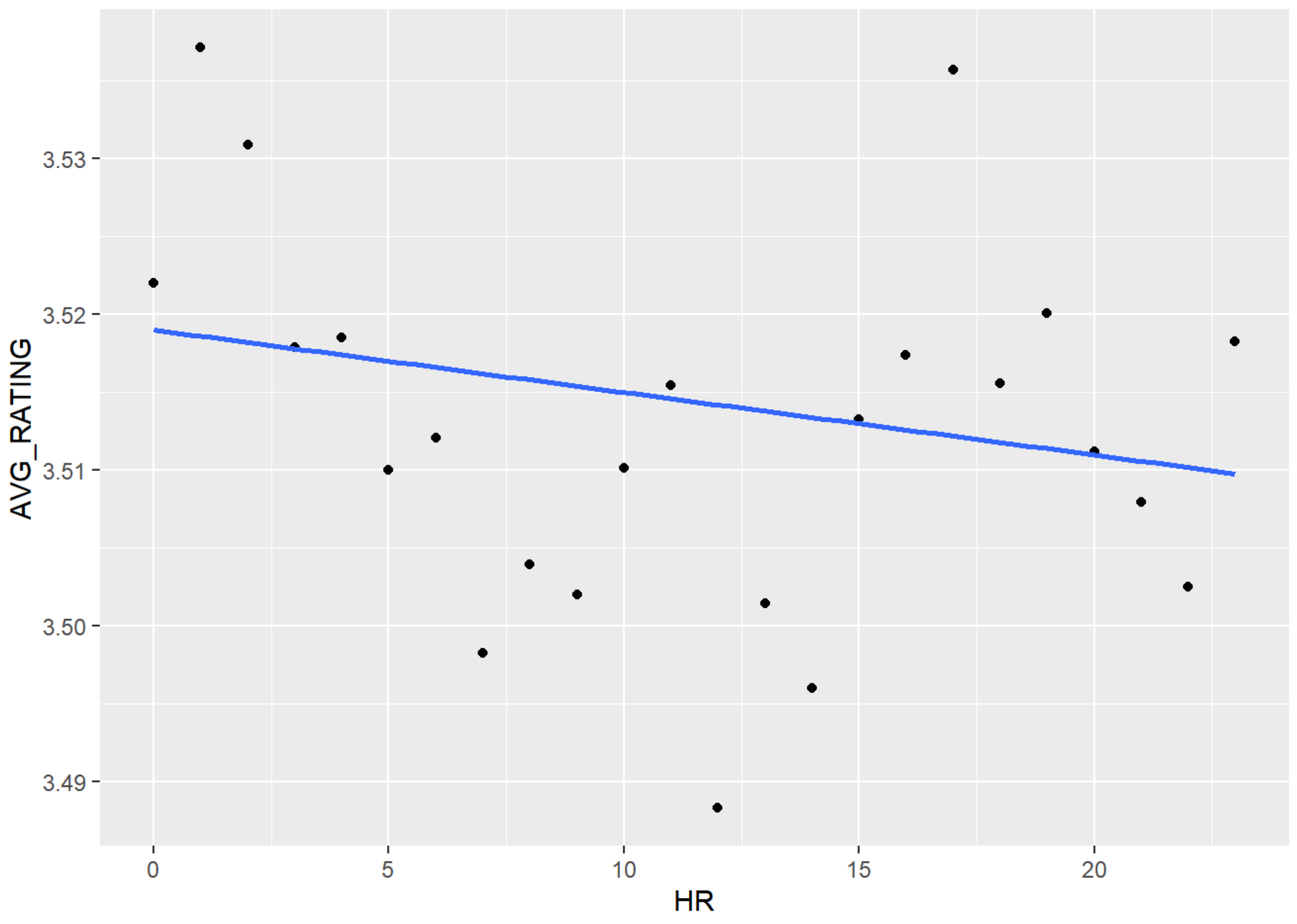


Here we see higher ratings in period Nov-Jan and lower the rest months, but not big differences.

```
train_set_sample%>%group_by(HR)%>%summarize(HR,AVG_RATING=mean(rating))%>%ggplot(aes(HR,AVG_R
ATING))+geom_point()+geom_smooth(method = "lm")
```

```
## `summarise()` regrouping output by 'HR' (override with `.groups` argument)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Here we do not see really affectation by the hour or rating to rating itself. So we will not test this variable in our models.
Another variable that we will use as predictor is genres. We think type affects rating. Lets find out how many unique categories genres we have.

```
length(unique(edx$genres))
```

```
## [1] 797
```

There are almost 800 categories and we have 9 millions samples. So no reason to break down genres, we will use it as it is.

BUILDING OUR MODELS - RESULTS

So we are ready to pass to build our model. Criterion is RMSE that we define here.

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

We start calculating the average rating.

```
mu <- mean(train_set$rating)
```

Calculate f_m (factor movie) on the training set.

```
movie_avgs <- train_set %>%group_by(movieId) %>% summarize(f_m = mean(rating - mu))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
predicted_ratings_f_m <- mu + test_set %>% left_join(movie_avgs, by='movieId') %>%.$f_m
```

Set mu any possible NA that could destroy our RMSE calculation (one is enough).

```
predicted_ratings_f_m[is.na(predicted_ratings_f_m)]<-mu
RMSE(predicted_ratings_f_m,test_set$rating)
```

```
## [1] 0.9441596
```

Add user factor

```
user_avgs <- train_set %>% left_join(movie_avgs, by='movieId') %>%group_by(userId) %>%
summarize(f_u = mean(rating - mu - f_m))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
predicted_ratings_f_u <- test_set %>%
left_join(movie_avgs, by='movieId') %>%
left_join(user_avgs, by='userId') %>%
mutate(pred = mu + f_m + f_u) %>%
.$pred

predicted_ratings_f_u[is.na(predicted_ratings_f_u)]<-mu
RMSE(predicted_ratings_f_u,test_set$rating)
```

```
## [1] 0.8659785
```

So here we are more on less on the level of the taught course userId+movieId and RMSE of 0.8659 Lets try go further trying other factors to see if they will improne our criterion.

Adding genres

```
genres_avgs <- train_set %>%
left_join(movie_avgs, by='movieId') %>%
left_join(user_avgs, by='userId') %>%
group_by(genres) %>%
summarize(f_g = mean(rating - mu - f_m - f_u))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
predicted_ratings_f_g <- test_set %>%
left_join(movie_avgs, by='movieId') %>%
left_join(user_avgs, by='userId') %>%
left_join(genres_avgs, by='genres') %>%
mutate(pred = mu + f_m + f_u + f_g) %>%
.$pred

predicted_ratings_f_g[is.na(predicted_ratings_f_g)]<-mu
RMSE(predicted_ratings_f_g,test_set$rating)
```

```
## [1] 0.8656067
```

So we have further improvement. We reached RMSE of 0.8656.

Add MA factor (movie age)

```
ma_avgs <- train_set %>%
left_join(movie_avgs, by='movieId') %>%
left_join(user_avgs, by='userId') %>%
left_join(genres_avgs, by='genres') %>%
group_by(MA) %>%
summarize(f_ma = mean(rating - mu - f_u - f_m-f_g))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
predicted_ratings_f_a <- test_set %>%
left_join(movie_avgs, by='movieId') %>%
left_join(user_avgs, by='userId') %>%
left_join(genres_avgs, by='genres') %>%
left_join(ma_avgs,by='MA') %>%
mutate(pred = mu + f_m + f_u + f_g+f_ma) %>%
.$pred


predicted_ratings_f_a[is.na(predicted_ratings_f_a)]<-mu
RMSE(predicted_ratings_f_a,test_set$rating)
```

```
## [1] 0.8651742
```

We reached RMSE 0.8651

Adding NRPU

```
nrpu_avgs <- train_set %>%
left_join(movie_avgs, by='movieId') %>%
left_join(user_avgs, by='userId') %>%
left_join(genres_avgs, by='genres') %>%
left_join(ma_avgs,by='MA') %>%
group_by(NRPU)%>%
summarize(f_nrpu = mean(rating - mu - f_u - f_m-f_g-f_ma))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
predicted_ratings_f_nrpu <- test_set %>%
left_join(movie_avgs, by='movieId') %>%
left_join(user_avgs, by='userId') %>%
left_join(genres_avgs, by='genres') %>%
left_join(ma_avgs,by='MA') %>%
left_join(nrpu_avgs,by='NRPU')%>%
mutate(pred = mu + f_m + f_u + f_g+f_ma+f_nrpu) %>%
.$pred

predicted_ratings_f_nrpu[is.na(predicted_ratings_f_nrpu)]<-mu

RMSE(predicted_ratings_f_nrpu,test_set$rating)
```

```
## [1] 0.901913
```

Get worse our results. We throw it away as factor.

Try NRPM

```
nrpm_avgs <- train_set %>%
left_join(movie_avgs, by='movieId') %>%
left_join(user_avgs, by='userId') %>%
left_join(genres_avgs, by='genres') %>%
left_join(ma_avgs,by='MA') %>%
group_by(NRPM)%>%
summarize(f_nrpm = mean(rating - mu - f_u - f_m-f_g-f_ma))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
predicted_ratings_f_nrpm <- test_set %>%
left_join(movie_avgs, by='movieId') %>%
left_join(user_avgs, by='userId') %>%
left_join(genres_avgs, by='genres') %>%
left_join(ma_avgs,by='MA') %>%
left_join(nrpm_avgs,by='NRPM')%>%
mutate(pred = mu + f_m + f_u + f_g+f_ma+f_nrpm) %>%
.$pred

predicted_ratings_f_nrpm[is.na(predicted_ratings_f_nrpm)]<-mu

RMSE(predicted_ratings_f_nrpm,test_set$rating)
```

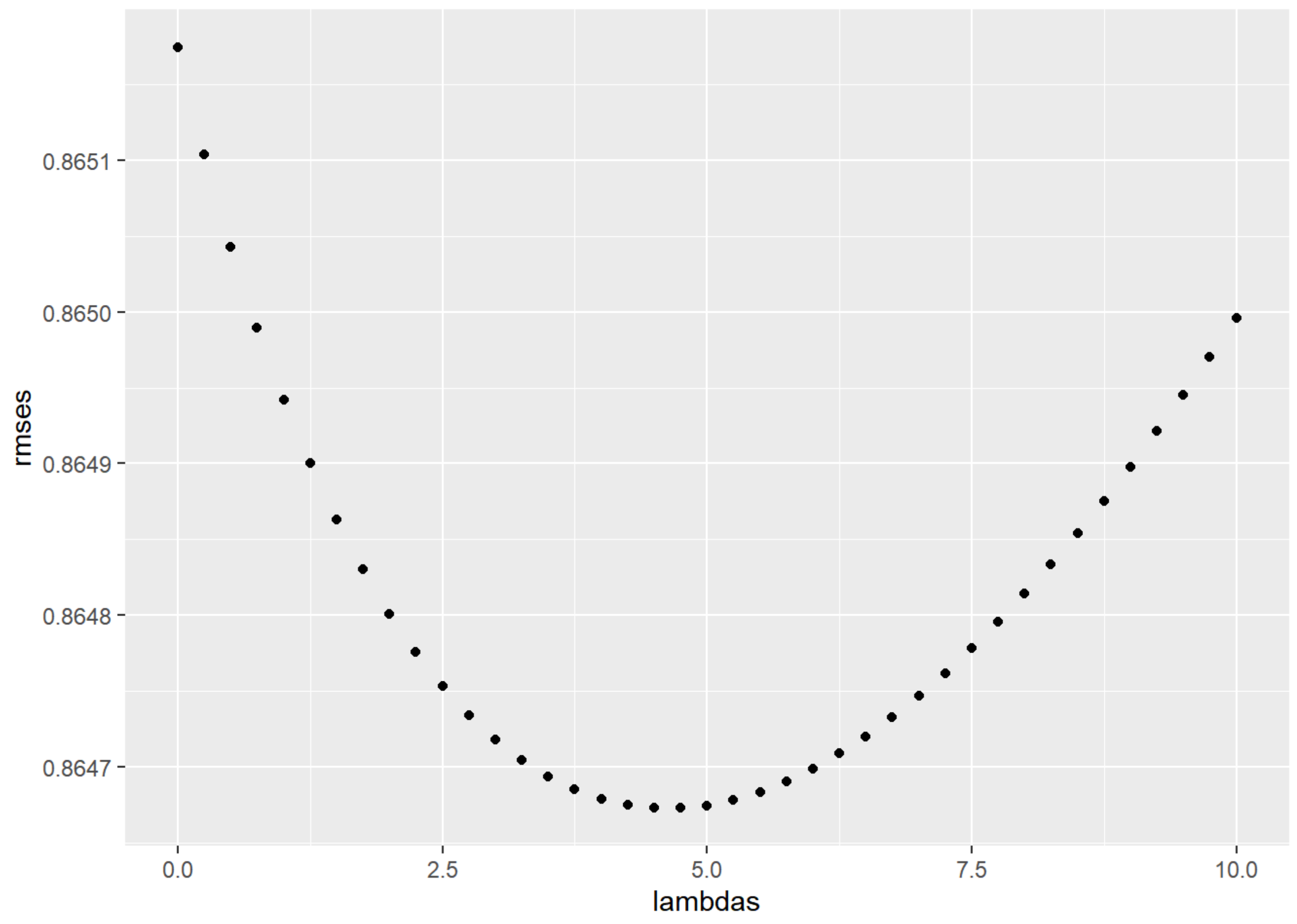
```
## [1] 0.8995502
```

Again the same happens. So we will remain with movieId, UserId, genres and MA (movie age).

The only that remains to try more is REGULARIZATION. We will apply it in all factors to suppress `noise'. Saying noise we mean for example in case of movieId, entries in our dataset of movies with very few ratings.

```
lambdas <- seq(0, 10, 0.25)
rmses <- sapply(lambdas, function(l){
  mu <- mean(train_set$rating)
  f_m <- train_set %>%
    group_by(movieId) %>%
    summarize(f_m = sum(rating - mu)/(n()+l),.groups='drop')
  f_u <- train_set %>%
    left_join(f_m, by="movieId") %>%
    group_by(userId) %>%
    summarize(f_u = sum(rating - f_m - mu)/(n()+l),.groups='drop')
  f_g <- train_set %>%
    left_join(f_m, by="movieId") %>%
    left_join(f_u, by="userId") %>%
    group_by(genres) %>%
    summarize(f_g= sum(rating - f_m - f_u-mu)/(n()+l),.groups='drop')
  f_a<- train_set %>%
    left_join(f_m, by="movieId") %>%
    left_join(f_u, by="userId") %>%
    left_join(f_g, by="genres") %>%
    group_by(MA) %>%
    summarize(f_a= sum(rating - f_m - f_u - f_g - mu)/(n()+l),.groups='drop')
  predicted_ratings <-
    test_set %>%
    left_join(f_m, by = "movieId") %>%
    left_join(f_u, by = "userId") %>%
    left_join(f_g, by = "genres") %>%
    left_join(f_a, by = "MA") %>%
    mutate(pred = mu + f_m + f_u + f_g + f_a) %>%
    .$pred
  predicted_ratings[is.na(predicted_ratings)]<-mu
  return(RMSE(predicted_ratings, test_set$rating))
})

qplot(lambdas, rmses)
```



From the plot is clear further improvement that brings us to the best possible RMSE that project asks.
The lamda that gives minimum RMSE and the minimum RMSE are:

```
lamda<-lambdas[which.min(rmses)]
min(rmses)
```

```
## [1] 0.8646729
```

```
lamda
```

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

So we have a perfect RMSE of 0.86467 with lamda just below 5. This value will be used for validation prediction.

As we have reached our final model it is time prepare the validation set. We have to bring it to the same form as train or test sets. Of course no need do the steps for NRPM or NRPU that finally we do not use.The same for HR or MR.

```
validation<-validation%>% mutate(Rating_date=as.Date(as.POSIXct(timestamp, origin = "1970-01-01") ))
validation<-validation%>% mutate(RY=as.numeric(format(Rating_date, '%Y')))
validation<-validation%>%mutate(MY=substr(title,nchar(title)-4,nchar(title)-1))
validation<- validation%>%mutate(MA=RY-as.numeric(MY))
validation$title<-NULL
validation$MY<-NULL
validation$RY<-NULL
validation$timestamp<-NULL
validation$Rating_date<-NULL
```

APLICATION OF FINAL MODEL TO VALIDATION SET

```
mu <- mean(edx$rating)
f_m <- train_set %>%
  group_by(movieId) %>%
  summarize(f_m = sum(rating - mu)/(n()+lamda),.groups='drop')
f_u <- train_set %>%
  left_join(f_m, by="movieId") %>%
  group_by(userId) %>%
  summarize(f_u = sum(rating - f_m - mu)/(n()+lamda),.groups='drop')
f_g <- train_set %>%
  left_join(f_m, by="movieId") %>%
  left_join(f_u, by="userId") %>%
  group_by(genres) %>%
  summarize(f_g= sum(rating - f_m - f_u-mu)/(n()+lamda),.groups='drop')
f_a<- train_set %>%
  left_join(f_m, by="movieId") %>%
  left_join(f_u, by="userId") %>%
  left_join(f_g, by="genres") %>%
  group_by(MA) %>%
  summarize(f_a= sum(rating - f_m - f_u - f_g - mu)/(n()+lamda),.groups='drop')
predicted_ratings <-
  validation %>%
  left_join(f_m, by = "movieId") %>%
  left_join(f_u, by = "userId") %>%
  left_join(f_g, by = "genres") %>%
  left_join(f_a, by = "MA") %>%
  mutate(pred = mu + f_m + f_u + f_g + f_a) %>%
  .$pred
predicted_ratings[is.na(predicted_ratings)]<-mu
RMSE(predicted_ratings, validation$rating)
```

```
## [1] 0.8644001
```

One note here, we used the whole edx as training here to take advance from the bigger pool of samples. No need work here with train_set as before.

CONCLUSIONS

We managed have an RMSE of 0.8644 on the validation set, using the known variables userId and movieId plus genres and another variable created named MA (movie age) that shows the difference of years between production and rating. Finally on top of them we applied reguralization in all these variables that suppress

‘noise’.
Limitations we faced were that adding more predictors is not always better. It is important their combination. Another one was transforming our datasets in order produce the experimental new variables. We saw that we had to apply the changes separately in train and test set and not on the whole edx set and then split it. Doing the last we had crash even with 8 or 12 GB ram.The HW is a great limitation in ML and here comes the possible future work.
Initially we tried apply other models using caret package like knn or rpart or even random forest but this was really impossible having sometimes to wait even 3 days without result. Or other times receiving memory exceptions and errosr. So would be really interesting try models like the ones mentioned but with such big dataset, HW like 8-12GB RAM and processor i5 is nothing. So yes would be a challenge using a much more powerfull machine applying other models on this dataset and see their results compared with the ones we have.