Recommender-System

Bilal Khaiar

11/15/2019

Introduction

Recommender systems are a subclass of information filtering systems that seeks to predict the "rating" or "preference" a user would give to an item. For more information click (here).

Recommender systems are an integral part of today's online markets and streaming services. Due to this huge demand, new techniques are constantly being developed and old ones are being optimized. It is one of the most widely used applications of Machine Learning.

This project is part of the Harvardx Data Science capstone course. The goal of the project is to build a recommender system that can predict movie ratings by different users for the Movielens dataset.

On the first approach, an attempt to optimize the model described in the book "Introduction to Data Science" by Rafael A. Irizarry will be made. The second approach involves Matrix factorization using the Recosystem package.

Methods

Libraries and Data

The following packages are needed to run the project's code.

```
library(tidyverse)
library(caret)
library(lubridate)
library(knitr)
library(recosystem)
library(data.table)
library(stringr)
```

Preparing the Data

The movielens dataset is downloaded and movies names and ratings are attached to it.

```
dl <- tempfile()
download.file("http://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

ratings <- fread(text = gsub("::", "\t", readLines(unzip(dl, "ml-10M100K/ratings.dat"))),
                 col.names = c("userId", "movieId", "rating", "timestamp"))

movies <- str_split_fixed(readLines(unzip(dl, "ml-10M100K/movies.dat")), "\\::", 3)
colnames(movies) <- c("movieId", "title", "genres")
movies <- as.data.frame(movies) %>% mutate(movieId = as.numeric(levels(movieId))[movieId],
                                           title = as.character(title),
                                           genres = as.character(genres))

movielens <- left_join(ratings, movies, by = "movieId")
rm(movies, ratings)
```

A Validation set will be created. It will be 10% of MovieLens data. this validation set will only be used for final assessment.

```
set.seed(1)
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.2, 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(test_index, temp, removed, movielens)
```

The loss function used to assess the accuracy of this system will be the Root Mean Square Error "RMSE".

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

Exploratory Analysis

In order to get insights into the dataset, some exploratory analysis will be made.

```
str(edx)
```

```
## 'data.frame':    8000076 obs. of  6 variables:
##  $ userId   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ movieId  : num  122 185 231 292 316 329 355 356 362 370 ...
##  $ rating   : num  5 5 5 5 5 5 5 5 5 5 ...
##  $ timestamp: int  838985046 838983525 838983392 838983421 838983392 838983392 838984474 838983653 83
##  $ title    : chr  "Boomerang (1992)" "Net, The (1995)" "Dumb & Dumber (1994)" "Outbreak (1995)" ...
##  $ genres   : chr  "Comedy|Romance" "Action|Crime|Thriller" "Comedy" "Action|Drama|Sci-Fi|Thriller"
```

The movieId variable is stored as a numeric vector. Integers are more memory efficient but they only hold whole number values. A test to check if all values are whole numbers will be made.

```
identical(round(edx$movieId, digits = 0), edx$movieId)
```

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

Since all values are whole numbers, the column can safely be converted into class integer without changing the actual values.

The genres variable seems to hold multiple genres, most likely ordered by relevance. By counting how many separators "|" are available in each genres observation and then adding 1, we can calculate how many parts does the genres variable have.

```
summary(str_count(edx$genres, pattern = "\\|") + 1)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   2.597   3.000   8.000
```

```
g_l <- length(unique(edx$genres))
g_l
```

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

The median genre is 3 parts long, they ranges from 1 to 8 parts and there are 797 unique types of genres. That can pose a challenge in prediction. Some genres might occur only once or twice, having a small sample size does not provide much information to the models.

Wrangling

The timestamp variable will be converted to a separate date and time columns to ease further analysis. The title variable holds the year each movie was made. Movie age in years at 2019 will calculated. The genres variable will be split into 5 different columns while keeping the original variable to study the effect of levels reduction on variability in the average rating. The movieId variable will be converted to an integer vector.

This chunk of code will generate warnings. Any values beyond the fifth part of the genres variable will be omitted.

```
edx <- edx %>%
  separate(col = genres, into = c("genre1","genre2","genre3","genre4", "genre5"),
           sep = "\\|", remove = FALSE) %>%
  mutate(movieId = as.integer(movieId),
          year = str_extract(title, pattern = "\\(\\d{4}\\)"),
         year = str_replace(year, "\\(", ""),
         year = str_replace(year, "\\)", ""),
         year = as.numeric(year),
         age = 2019 - year,
         date = as_datetime(timestamp)) %>%
  separate(col = date, into = c("date", "time"), sep = "\\s")
```

In order to reduce the number of levels thus improving prediction and generalizing the model, an attempt to compare the standard deviation and levels of using different parts of the column will be made. The goal is to reduce levels as much as possible while minimizing the effect on rating variability.

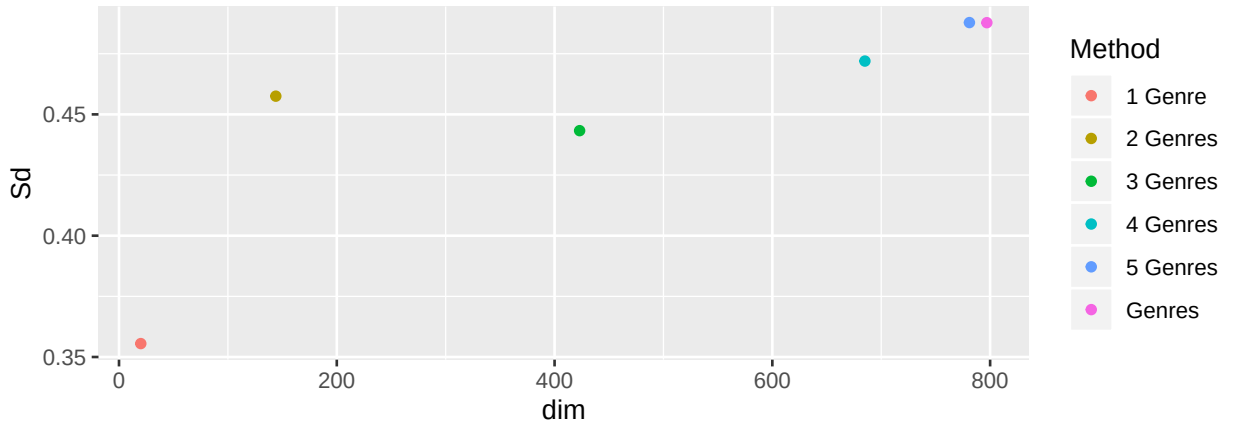
```
gen <- edx %>% select(genres, rating) %>%
  group_by(genres) %>%
  summarise(avg = mean(rating))
df <- tibble(Method = "Genres",
                 Sd = sd(gen$avg), dim = length(gen$avg))

gen <- edx %>% select(genre1, genre2, genre3, genre4, genre5, rating) %>%
  group_by(genre1, genre2, genre3, genre4, genre5) %>%
  summarise(avg = mean(rating))
df <- rbind(df,tibble(Method = "5 Genres",
                          Sd = sd(gen$avg), dim = length(gen$avg)))

gen <- edx %>% select(genre1, genre2, genre3, genre4, rating) %>%
  group_by(genre1, genre2, genre3, genre4) %>%
  summarise(avg = mean(rating))
df <- rbind(df,tibble(Method = "4 Genres",
                          Sd = sd(gen$avg), dim = length(gen$avg)))

gen <- edx %>% select(genre1, genre2, genre3, rating) %>%
  group_by(genre1, genre2, genre3) %>%
  summarise(avg = mean(rating))
df <- rbind(df,tibble(Method = "3 Genres",
                          Sd = sd(gen$avg), dim = length(gen$avg)))
```

```
gen <- edx %>% select(genre1, genre2, rating) %>%
  group_by(genre1, genre2) %>%
  summarise(avg = mean(rating))
df <- rbind(df,tibble(Method = "2 Genres",
                          Sd = sd(gen$avg), dim = length(gen$avg)))

gen <- edx %>% select(genre1, rating) %>%
  group_by(genre1) %>%
  summarise(avg = mean(rating))
df <- rbind(df,tibble(Method = "1 Genre",
                          Sd = sd(gen$avg), dim = length(gen$avg)))
kable(df)
```

Method	Sd	dim
Genres	0.4877953	797
5 Genres	0.4878419	781
4 Genres	0.4719675	685
3 Genres	0.4432892	423
2 Genres	0.4575069	144
1 Genre	0.3555259	20

Both dimensions and the standard deviation of ratings are reduced when using a smaller part of the genres variable.

```
df %>% ggplot(aes(dim, Sd, color = Method)) + geom_point()
```



When using only the first 2 parts in the genres variable, percentage of dimension reduction and loss in variability "measured in standard deviation" is calculated this way.

```
paste("dimension reduciton ", round(100 - df[5,3]/df[1,3]*100), "%")
paste("loss in variablity ", round(100 - df[5,2]/df[1,2]*100),"%")
```

```
## [1] "dimension reduciton  82 %"
## [1] "loss in variablity  6 %"
```

Only the first 2 parts of the genres variable will be kept.

```
edx <- edx %>% select(-c(genre3,genre4, genre5, genres)) %>%
  unite(genres,genre1, genre2, sep = "-", na.rm = TRUE)
```

Building Models

Edx will be split into test and train sets. This test set will be used to assess and compare models accuracy.

```
set.seed(2)
test_index <- createDataPartition(y = edx$rating, times = 1,
                                  p = 0.2, list = FALSE)
train_set <- edx[-test_index,]
temp <- edx[test_index,]

test_set <- temp %>%
  semi_join(train_set, by = "movieId") %>%
  semi_join(train_set, by = "userId")

removed <- anti_join(temp, test_set)
train_set <- rbind(train_set, removed)

rm(test_index, temp, removed)
```

In order to save RAM space, the wrangled edx set and the validation set will be stored in the hard drive "for final validation" and removed from RAM for now.

```
# save edx and validation sets in HDD
save(edx, validation, file = "edx_val.rda")
# remove edx and validation sets from RAM
rm(edx, validation)
```

Model 1

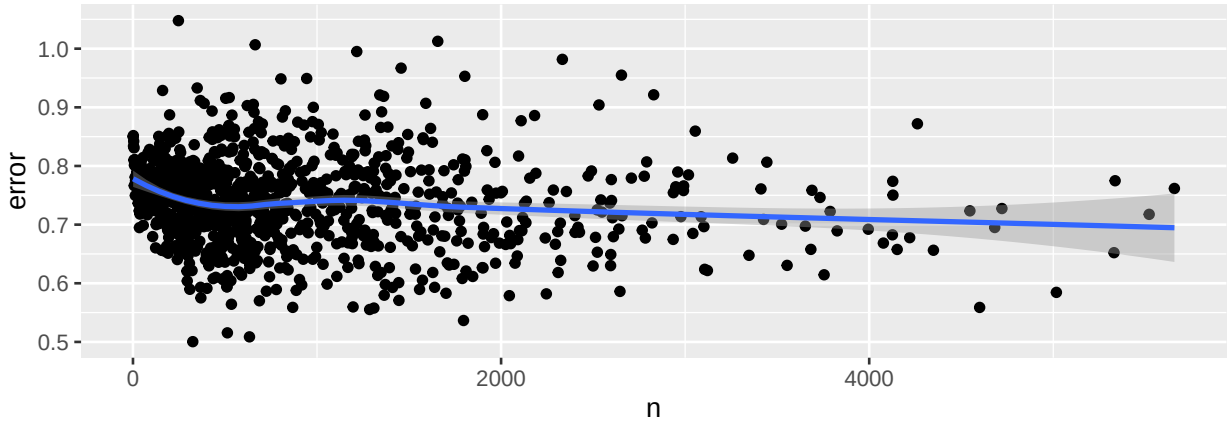
Some movies have higher rating than others regardless of other parameters. the first model will predict each movie's average rating.

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

# Calculating the standardized average for each movie.
movie_avgs <- train_set %>%
    group_by(movieId) %>%
    summarize(b_i = mean(rating - mu))

predicted_ratings <- mu + test_set %>%
    left_join(movie_avgs, by='movieId') %>%
    .$b_i

model_1 <- RMSE(predicted_ratings, test_set$rating)


rmse_results <- tibble(Method="Movie Effect Model",
                                  RMSE = model_1)

kable(rmse_results)
```

Method	RMSE
Movie Effect Model	0.9434262

Exploring each model's worse predictions can help optimize the model.
An error variable will be created. The relation between the number of times each movie was rated and the amount of error will be studied.

```
t <- test_set %>%
  mutate(pred = predicted_ratings,error = abs(rating - pred)) %>%
  group_by(title) %>% mutate(n = n()) %>% ungroup()
t %>% group_by(n) %>%
  summarise(error = mean(error)) %>%
  arrange(desc(error))%>% ggplot(aes(n, error)) +
  geom_point() + geom_smooth(method = "loess")
```



The plot shows that the mean error is lower for movies with a higher number of ratings.

```
low_n_rmse <- t %>% filter(n < 20) %>% summarise(RMSE_low_n = sqrt(mean(error^2)))
high_n_rmse <- t %>% filter(n > 20) %>% summarise(RMSE_high_n = sqrt(mean(error^2)))
print(c(low_n_rmse,high_n_rmse))
```

```
## $RMSE_low_n
## [1] 1.024679
##
## $RMSE_high_n
## [1] 0.9415948
```

Model accuracy is better for movies with over 20 ratings. 20 was just a random number. Optimization of regularization parameters is the next step.

Regularization

Regularization penalises movies with a small number of ratings and moves their predicted rating closer to the general average. this can be done by either using a cut off "and later using an ensemble or a different model to predict this subgroup", like the previous example, or by using weights to penalize movies with small n numbers. The second approach will be used in the next model due to its ability to include all data.

Tuning regularization parameters will done by splitting the train set further into a tuning set "for cross validation" and a slightly smaller train set. Using the test set to tune any parameters will cause overfitting and give RMSE results that are over confident. Both new sets will be used only to tune the regularization parameter.

```
set.seed(3)
tune_index <- createDataPartition(train_set$rating, times = 1,
```

```
                                    p = 0.2, list = FALSE)
train_set2 <- train_set[-tune_index,]
temp <- train_set[tune_index,]

tune_set <- temp %>%
  semi_join(train_set2, by = "movieId") %>%
  semi_join(train_set2, by = "userId")

removed <- anti_join(temp, tune_set)
train_set2 <- rbind(train_set2, removed)

rm(tune_index, temp, removed)
```

Model 2

This model will use the average movie rating while penalising movies with a small number of ratings.

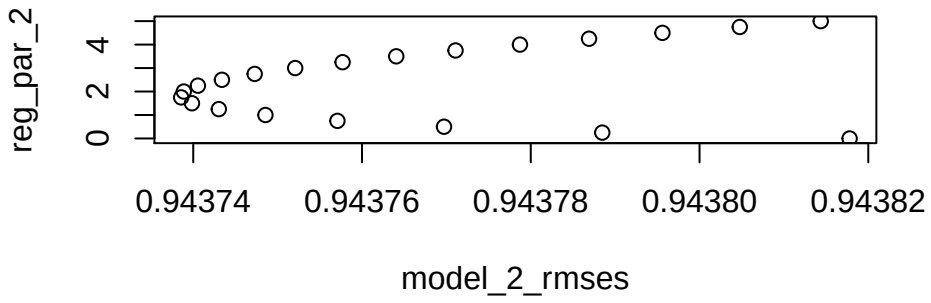
In order to check most possible values, the parameter will be found by using large intervals at first "seq(0, 100, 10)" and then narrowing down to smaller intervals until finding the best parameter. This method was inferred from Michael Nielsen's online book "Neural Networks and Deep Learning" (here).

A loop will be created to find the parameter which gives the lowest RMSE on the tuning set "cross validation". After a couple of attempts, the possible values for the parameter were narrowed down to a smaller list.

```
reg_par_2 <- seq(0, 5, 0.25)
# This loop will try different parameters and outputs RSMEs
model_2_rmses <- lapply(reg_par_2, function(p){
  movie_avgs <- train_set2 %>%
    group_by(movieId) %>%
    summarize(b_m = sum(rating - mu)/(n()+p))

  predicted_ratings <- mu + tune_set %>%
    left_join(movie_avgs, by='movieId') %>%
    .$b_m
  RMSE(predicted_ratings, tune_set$rating)
})

# The regularization parameter with the lowest RMSE is.
p_m <- reg_par_2[which.min(model_2_rmses)]
p_m
```

```
## [1] 1.75
```

```
plot(model_2_rmses, reg_par_2)
```



Regularized movieID based model.

```
r_movie_avgs <- train_set %>%
  group_by(movieId) %>%
  summarize(r_b_m = sum(rating - mu)/(n()+p_m))

predicted_ratings <- mu + test_set %>%
  left_join(r_movie_avgs, by='movieId') %>%
  .$r_b_m

model_2 <- RMSE(predicted_ratings, test_set$rating)



rmse_results <- rbind(rmse_results, tibble(Method="Movie Effect Model Regularized",
                                RMSE = model_2))

kable(rmse_results)
```

Method	RMSE
Movie Effect Model	0.9434262
Movie Effect Model Regularized	0.9433413

The accuracy of the model is slightly improved. The same approach will be used with other variables

Model 3

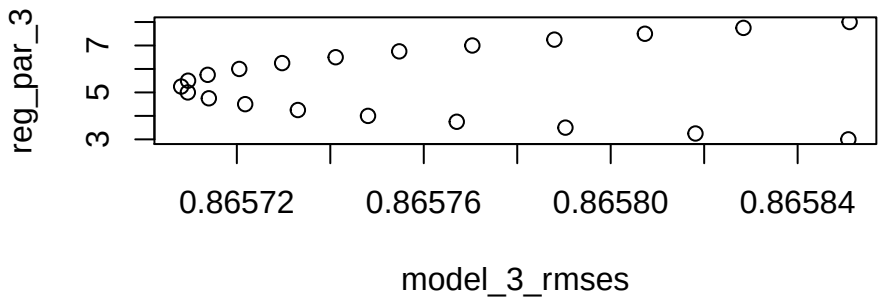
It will be assumed that different users tend to give a different average rate for movies they rate, an attempt to add that to the model will be made. A regularized model will be built as the effect of small sample size on model accuracy has been already shown in the previous model.

```
reg_par_3 <- seq(3, 8, 0.25)

model_3_rmses <- lapply(reg_par_3, function(p){

  user_avgs <- train_set2 %>%
```

```
    left_join(r_movie_avgs, by='movieId') %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - mu - r_b_m)/(n()+p))

  predicted_ratings <- tune_set %>%
    left_join(r_movie_avgs, by='movieId') %>%
    left_join(user_avgs, by='userId') %>%
    mutate(pred = mu + r_b_m + b_u) %>%
    .$pred
  RMSE(predicted_ratings, tune_set$rating)
})

# The best parameter is.
p_u <- reg_par_3[which.min(model_3_rmses)]
p_u
```

```
## [1] 5.25
```

```
plot(model_3_rmses, reg_par_3)
```



Regularized movieID + userID based model.

```
r_user_avgs <- train_set %>%
  left_join(r_movie_avgs, by='movieId') %>%
  group_by(userId) %>%
  summarize(r_b_u = sum(rating - mu - r_b_m)/(n()+p_u))

predicted_ratings <-
  test_set %>%
  left_join(r_movie_avgs, by = "movieId") %>%
  left_join(r_user_avgs, by = "userId") %>%
  mutate(pred = mu + r_b_m + r_b_u) %>%
  .$pred

model_3 <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- rbind(rmse_results,
```

```
                      tibble(Method="Movie + User Effect Model Regularized",
                             RMSE = model_3))

kable(rmse_results)
```

Method	RMSE
Movie Effect Model	0.9434262
Movie Effect Model Regularized	0.9433413
Movie + User Effect Model Regularized	0.8663777

By adding averages of each user, the accuracy of the model got significantly higher.

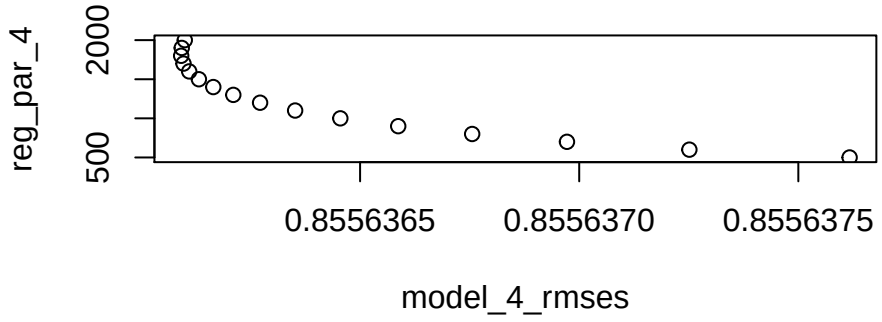
Model 4

This model will add the average of each genre.

```
reg_par_4 <- seq(500, 2000, 100)

model_4_rmses <- lapply(reg_par_4, function(p){

  genres_avgs <- train_set2 %>%
    left_join(r_movie_avgs, by='movieId') %>%
    left_join(r_user_avgs, by='userId') %>%
    group_by(genres) %>%
    summarize(b_g = sum(rating - mu - r_b_m - r_b_u)/(n()+p))

  predicted_ratings <- tune_set %>%
    left_join(r_movie_avgs, by='movieId') %>%
    left_join(r_user_avgs, by='userId') %>%
    left_join(genres_avgs, by='genres') %>%
    mutate(pred = mu + r_b_m + r_b_u + b_g) %>%
    .$pred
  RMSE(predicted_ratings, tune_set$rating)
})

# The best parameter is.
p_g <- reg_par_4[which.min(model_4_rmses)]
p_g
```

```
## [1] 1800
```

```
plot(model_4_rmses, reg_par_4)
```



Although the regularization parameter list can be narrowed down further, the plot shows that change in rmse is so small and not significant.

Regularized movieID + userID + genres based model.

```
r_genres_avgs <- train_set %>%
  left_join(r_movie_avgs, by='movieId') %>%
  left_join(r_user_avgs, by='userId') %>%
  group_by(genres) %>%
  summarize(r_b_g = sum(rating - mu - r_b_m - r_b_u)/(n()+p_g))

predicted_ratings <-
  test_set %>%
  left_join(r_movie_avgs, by = "movieId") %>%
  left_join(r_user_avgs, by = "userId") %>%
  left_join(r_genres_avgs, by = "genres") %>%
  mutate(pred = mu + r_b_m + r_b_u + r_b_g) %>%
  .$pred

model_4 <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- rbind(rmse_results,
                      tibble(Method="Movie + User + Genres Effect Model Regularized",
                             RMSE = model_4))

kable(rmse_results)
```

Method	RMSE
Movie Effect Model	0.9434262
Movie Effect Model Regularized	0.9433413
Movie + User Effect Model Regularized	0.8663777
Movie + User + Genres Effect Model Regularized	0.8662241

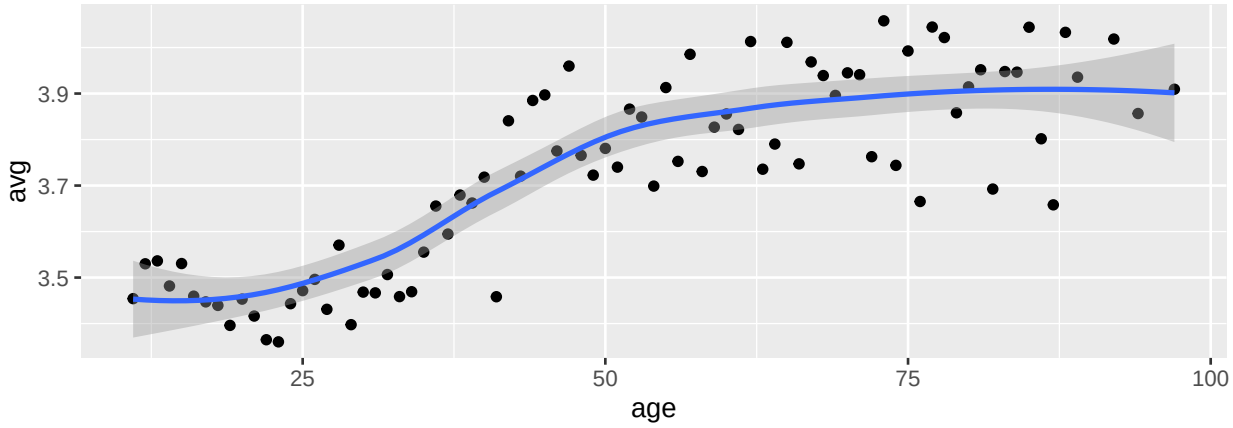
Model 5

On this model, the age variable will be introduced.

```
t <- train_set %>% select(age, rating) %>%
  group_by(age) %>% summarise(n = n(),avg = mean(rating)) %>% filter(n > 1000)

sd(t$avg)
```

```
## [1] 0.2093548
```

```
t %>% ggplot(aes(age,avg)) + geom_point() + geom_smooth(method = "loess")
```

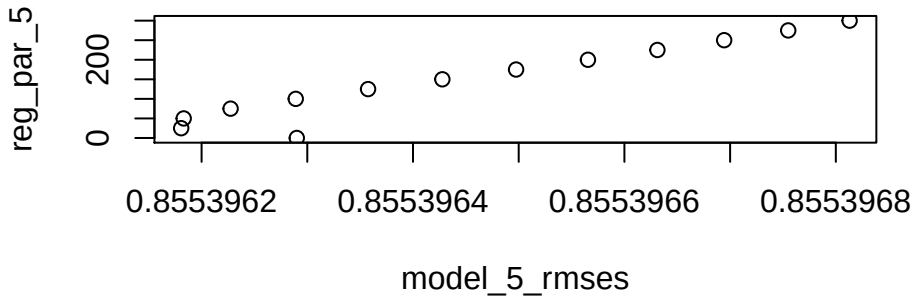


Older movies have higher average ratings than newer ones. Model 5 is going to try to improve accuracy by incorporating age averages.

```
reg_par_5 <- seq(0, 300,25)

model_5_rmses <- lapply(reg_par_5, function(p){

  age_avgs <- train_set2 %>%
    left_join(r_movie_avgs, by='movieId') %>%
    left_join(r_user_avgs, by='userId') %>%
    left_join(r_genres_avgs, by='genres') %>%
    group_by(age) %>%
    summarize(b_a = sum(rating - mu - r_b_m - r_b_u - r_b_g)/(n()+p))

  predicted_ratings <- tune_set %>%
    left_join(r_movie_avgs, by='movieId') %>%
    left_join(r_user_avgs, by='userId') %>%
    left_join(r_genres_avgs, by='genres') %>%
    left_join(age_avgs, by='age') %>%
    mutate(pred = mu + r_b_m + r_b_u + r_b_g + b_a) %>%
    .$pred
  RMSE(predicted_ratings, tune_set$rating)
})

# The best parameter is.
p_a <- reg_par_5[which.min(model_5_rmses)]
p_a
```

```
## [1] 25
```

```
plot(model_5_rmses, reg_par_5)
```



Regularized movieID + userID + genres + age based model.

```
r_age_avgs <- train_set %>%
  left_join(r_movie_avgs, by='movieId') %>%
  left_join(r_user_avgs, by='userId') %>%
  left_join(r_genres_avgs, by='genres') %>%
  group_by(age) %>%
  summarize(r_b_a = sum(rating - mu - r_b_m - r_b_u - r_b_g)/(n()+p_a))

predicted_ratings <-
  test_set %>%
  left_join(r_movie_avgs, by = "movieId") %>%
  left_join(r_user_avgs, by = "userId") %>%
  left_join(r_genres_avgs, by = "genres") %>%
  left_join(r_age_avgs, by = "age") %>%
  mutate(pred = mu + r_b_m + r_b_u + r_b_g + r_b_a) %>%
  .$pred

model_5 <- RMSE(predicted_ratings, test_set$rating)

rmse_results <- rbind(rmse_results,
                      tibble(Method="Movie + User + Genres + Age Effect Model Regularized",
                             RMSE = model_5))

kable(rmse_results)
```

Method	RMSE
Movie Effect Model	0.9434262
Movie Effect Model Regularized	0.9433413
Movie + User Effect Model Regularized	0.8663777
Movie + User + Genres Effect Model Regularized	0.8662241
Movie + User + Genres + Age Effect Model Regularized	0.8660090

Another slight improvement over the previous model but it is not good enough for the purpose of this project. On the next model, Matrix Factorization will be used.

Recosystem

Recosystem is a recommender system that uses matrix factorization. It creates a rating matrix and then approximates it by the dot products of 2 lower dimensional matrices P and Q. P holds k number of factors for user Ids and Q does the same for Movie Id. The concept is related to Singular Value Decomposition, but it works for sparse matrices. Every user rates a certain number of movies, that leaves the rating matrix sparse "has many NA values for movies not rated by specific users. Another approach to factorise sparse matrices is to substitute missing values with an adjusted average. This approach will not be used in this project.

Each of the two matrices have two penalty parameters named costp_l1, costp_l2, costq_l1 and costq_l2. for more information please check package vignettes. (here)

Furthermore, this package offers an option of working with data stored in the hard drive thus reducing needed RAM space. It is also a C++ wrapper, that offers the ability to compute on multiple threads in parallel. Although this can highly increase computation speed and save time, it will be avoided in the training process as it can reduce reproducibility of the code even if the seed was set. It will be used in tuning process as no effect on reproducibility was noticed.

First, all objects will be removed from RAM to clear space for model 6 except the rmse_result, test_set and train_set.

```
rm(list=ls()[! ls() %in% c("rmse_results","train_set","test_set")])
```

Transformation and Data links.

Data transformation is needed in order to fit the package input criteria. The package requires 2 dataframes. The train set should have 2 integer columns for user and item Ids, one numeric column for ratings. The test set only needs 2 integer columns.

```
r_train_set <- train_set %>% select(userId, movieId, rating)
r_test_set <- test_set %>% select(userId, movieId)
```

The data will be stored in the hard drive to minimize RAM usage. Data links will be created and the binding function will be names.

```
write.table(r_train_set , file = "r_train_set.txt" , sep = " ",
            row.names = FALSE, col.names = FALSE)
write.table(r_test_set, file = "r_test_set.txt" , sep = " ",
            row.names = FALSE, col.names = FALSE)

# Data links
r_train_set <- data_file("r_train_set.txt", package = "recosystem", index1 = TRUE)
r_test_set <- data_file("r_test_set.txt",  package = "recosystem", index1 = TRUE)

# Binding function will be used in tuning, training and prediction process.
r <- Reco()
```

Tuning

Warning : Tuning process can take a very long time to run. By using the same concept of narrowing down on regularization parameters, those tuning options where chosen. The process started by using the default options of the function r$tune(). For more information, check package documentation. (here)

```
set.seed(4)
r_tune <- r$tune(r_train_set, opts = list(dim = c(20L, 25L),
                                costp_l1 = 0,
                                costp_l2 = c(0.01,0.1),
                                costq_l1 =0,
```

```
                                costq_l2 = c(0.01,0.1),
                                lrate = 0.1,
                                nthread = 4,
                                niter = 20))
```

Training

Tuning parameters with the least RMSE will be used to train the model.

```
set.seed(5)
r_train <- r$train(r_train_set, opts = c(r_tune$min, niter = 30))
```

Predicting

The r$predict function will use the trained model to predict ratings on the test set

```
pred <- r$predict(r_test_set, out_memory())


model_6 <- RMSE(test_set$rating, pred)


rmse_results <- rbind(rmse_results,
                      tibble(Method = "Recosystem",
                                 RMSE = model_6))

kable(rmse_results)
```

Method	RMSE
Movie Effect Model	0.9434262
Movie Effect Model Regularized	0.9433413
Movie + User Effect Model Regularized	0.8663777
Movie + User + Genres Effect Model Regularized	0.8662241
Movie + User + Genres + Age Effect Model Regularized	0.8660090
Recosystem	0.7959724

Results

The 6th Model had the least RMSE. Matrix factorization is clearly a better approach when building recommender systems. In addition to that, the Recosystem package has the advantage of parallel computing which can save time when building models.

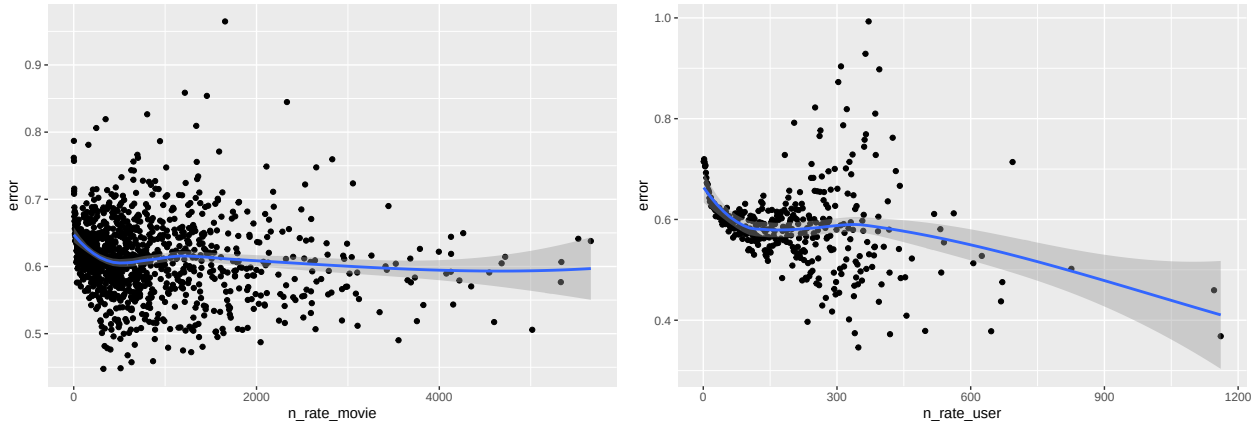
Further analysis will be made to check how the model works for subgroups of the data. An error variable will be created.

```
test_set$pred <- pred
test_set$error <- abs(test_set$pred - test_set$rating)
```

The effect of the number of ratings per user and per movie on the accuracy of model 6 will be explored

```
# plot error against the number of times each movie was rated and the number of times each user rated.
test_set %>% group_by(movieId) %>% mutate(n_rate_movie = n()) %>%
  ungroup() %>% group_by(n_rate_movie) %>% summarise(error = mean(error)) %>%
  arrange(desc(n_rate_movie))%>% ggplot(aes(n_rate_movie, error)) +
```

```
  geom_point() + geom_smooth(method = "loess")

test_set %>% group_by(userId) %>% mutate(n_rate_user = n()) %>% ungroup() %>%
  group_by(n_rate_user) %>% summarise(error = mean(error)) %>%
  arrange(desc(n_rate_user))%>% ggplot(aes(n_rate_user, error)) +
  geom_point() + geom_smooth(method = "loess")
```



The amount of error in predictions decreases for users with a high number of ratings. On the other hand, the number of ratings per movie doesn't change errors as much.

Future work

Model 6 can probably be improved by optimizing predictions for users with a small number of ratings. A model that depends on data from items instead of users might improve predictions for users with low ratings.

Project Validation

Model 6 RSME is relatively good for the purpose of this project so no further optimisation will be attempted. The final model will be tested on the validation set for project submission.

All object not needed will be removed from RAM.

```
rm(list=ls()[! ls() %in% c("r_tune","r")])
```

Edx and Validation sets will be loaded.

```
load("edx_val.rda")
```

Both sets will be transformed to fit package criteria.

```
r_edx <- edx %>% select(userId, movieId, rating)

r_validation <- validation %>% mutate(movieId = as.integer(movieId)) %>%
  select(userId, movieId)

# Write transformed sets to HDD
write.table(r_edx , file = "r_edx.txt" , sep = " ",
            row.names = FALSE, col.names = FALSE)
write.table(r_validation, file = "r_validation.txt" , sep = " ",
            row.names = FALSE, col.names = FALSE)
```

```r
# Create links
r_edx <- data_file("r_edx.txt", package = "recosystem", index1 = TRUE)
r_validation <- data_file("r_validation.txt",  package = "recosystem", index1 = TRUE)

# remove edx and keep validation set to calculate final RMSE
rm(edx)
```

The tuning step will not be repeated, the same parameters from testing will be used.

```r
set.seed(6)
r_train <- r$train(r_edx, opts = c(r_tune$min, niter = 30))
```

```r
pred <- r$predict(r_validation, out_memory())
RMSE(validation$rating, pred)
```

```
## [1] 0.7873669
```

Conclusion

The Recosystem package can build a better model in a shorter time by using matrix factorization. In addition, it supports parallel computing which can further reduce time.

Working on a huge dataset with limited computing power can show how important it is to try codes on random samples first and then apply them on the actual datasets.

Optimization can be an infinite task that never ends. Setting up a clear goal is one of the most important aspects of data science projects. Another aspect is setting a clear timeframe and getting the best results possible in this specific period.

References

1- Introduction to Data Science by Rafael A. Irizarry (here).

2- Neural Networks and Deep Learning by Michael Nielsen (here).

3- Recommender System Using Parallel Matrix Factorization by Yixuan Qiu (here).