MovieLens 10M Project

Wagner Rosa

#Overview In this project, we will analyse the Movie Lens data set that, in our case, will be consisted by a 10 million movie ratings and a 100,000 tag applications applied to 10,000 movies by 72,000 users. This orignal set was released in January 2009 and it serves as a stable benchmark for the development of recommendation systems. The goal of this project is to propose a recommendation model with the lowest root mean square error (RMSE) possible, which can be translated as a "good recommendation system". For this puporse, we will taking the following steps: - Perform an exploratory data analysis (EDA) of the data set - Propose and test classification models - Report the best model with RMSE

Now, let's create our data sets.

#Create test and validation sets In order to perform our analyses, we need to download the data either by using this link or by running the following code (provided by EDX).

```
################################
# Create edx set, validation set
################################

# Note: this process could take a couple of minutes

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

```
## Loading required package: tidyverse
```

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

```
## <U+2713> ggplot2 3.2.1     <U+2713> purrr   0.3.3
## <U+2713> tibble  2.1.3     <U+2713> dplyr   0.8.3
## <U+2713> tidyr   1.0.0     <U+2713> stringr 1.4.0
## <U+2713> readr   1.3.1     <U+2713> forcats 0.4.0
```

```
## -- 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.org")
```

```
## 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
```

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

```
## Loading required package: recosystem
```

```
# MovieLens 10M dataset:
# 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")
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")

# 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
```

```
# if using R 3.5 or earlier, use `set.seed(1)` instead
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)
```

After running the code above, we will have the 10M data set downloaded and, also, a validation set will be generated to be used in the future modeling.

#EDA ##Dimensions and properties Here, we are going to look if the data set is correctly downloaded and, at the same time, check the dimension of the data.

```
glimpse(edx)
```

```
## Observations: 9,000,055
## Variables: 6
## $ userId    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,...
## $ movieId   <dbl> 122, 185, 292, 316, 329, 355, 356, 362, 364, 370, 377, 420,...
## $ rating    <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,...
## $ timestamp <int> 838985046, 838983525, 838983421, 838983392, 838983392, 8389...
## $ title     <chr> "Boomerang (1992)", "Net, The (1995)", "Outbreak (1995)", "...
## $ genres    <chr> "Comedy|Romance", "Action|Crime|Thriller", "Action|Drama|Sc...
```

```
glimpse(validation)
```

```
## Observations: 999,999
## Variables: 6
## $ userId    <int> 1, 1, 1, 2, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5, 5, 5, 5, 5, 5, 5,...
## $ movieId   <dbl> 231, 480, 586, 151, 858, 1544, 590, 4995, 34, 432, 434, 85,...
## $ rating    <dbl> 5.0, 5.0, 5.0, 3.0, 2.0, 3.0, 3.5, 4.5, 5.0, 3.0, 3.0, 3.0,...
## $ timestamp <int> 838983392, 838983653, 838984068, 868246450, 868245645, 8682...
## $ title     <chr> "Dumb & Dumber (1994)", "Jurassic Park (1993)", "Home Alone...
## $ genres    <chr> "Comedy", "Action|Adventure|Sci-Fi|Thriller", "Children|Com...
```

We can see that we have successfully downloaded our data and we have created 2 data sets: *edx* (for training) and *validation* (for validation).

##Number of different movies in the dataset In this part, we'll check how many unique movies are included in the data set.

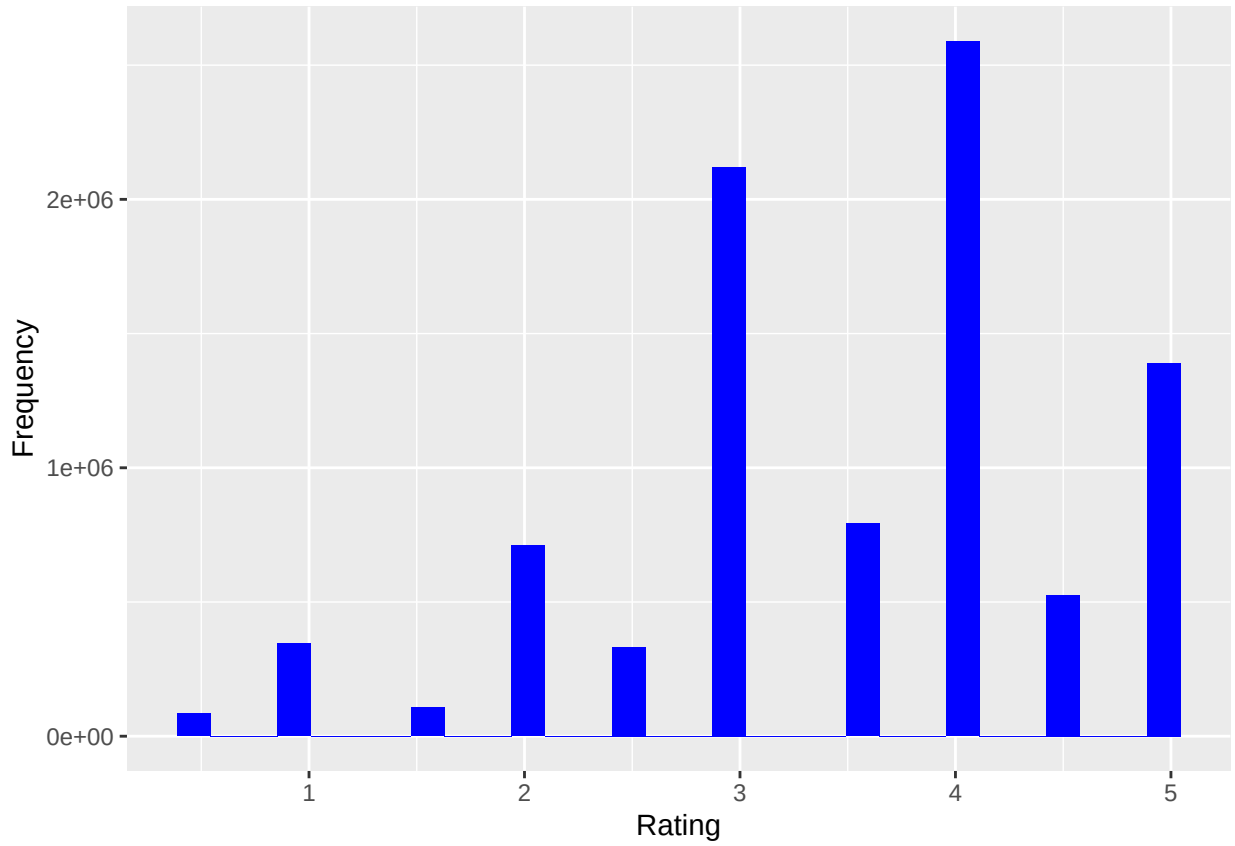
```
length(unique(edx$movieId))
```

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

##Ratings Now, let's look on how the ratings are distributed by making a hystogram of the *edx* data set

```
qplot(edx$rating,
      geom = "histogram",
      xlab = "Rating",
      ylab = "Frequency",
      fill = I("blue"))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



In general, users have a tendency to rate using whole stars rather than half star ratings, being the latest less common (e.g., there are fewer ratings of 3.5 than there are ratings of 3 or 4, etc.).

Looking for the five most given ratings in order from most to least.

```
edx %>% group_by(rating) %>%
  summarise(number = n()) %>%
  arrange(desc(number)) %>%
  top_n(5)
```

```
## Selecting by number
```

```
## # A tibble: 5 x 2
##   rating  number
##    <dbl>   <int>
## 1    4   2588430
## 2    3   2121240
## 3    5   1390114
## 4    3.5  791624
## 5    2    711422
```

##Movie with greatest number of ratings Checking which movies have the most number of ratings overall.

```
edx %>% group_by(title) %>%
  select(rating) %>%
  summarise(number = n()) %>%
  arrange(desc(number))
```

```
## Adding missing grouping variables: `title`
```

```
## # A tibble: 10,676 x 2
##    title                                                     number
##    <chr>                                                      <int>
##  1 Pulp Fiction (1994)                                        31362
##  2 Forrest Gump (1994)                                        31079
##  3 Silence of the Lambs, The (1991)                           30382
##  4 Jurassic Park (1993)                                       29360
##  5 Shawshank Redemption, The (1994)                           28015
##  6 Braveheart (1995)                                          26212
##  7 Fugitive, The (1993)                                       25998
##  8 Terminator 2: Judgment Day (1991)                          25984
##  9 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977)  25672
## 10 Apollo 13 (1995)                                           24284
## # ... with 10,666 more rows
```

Let's see what genres (by rating) are more popular among the users

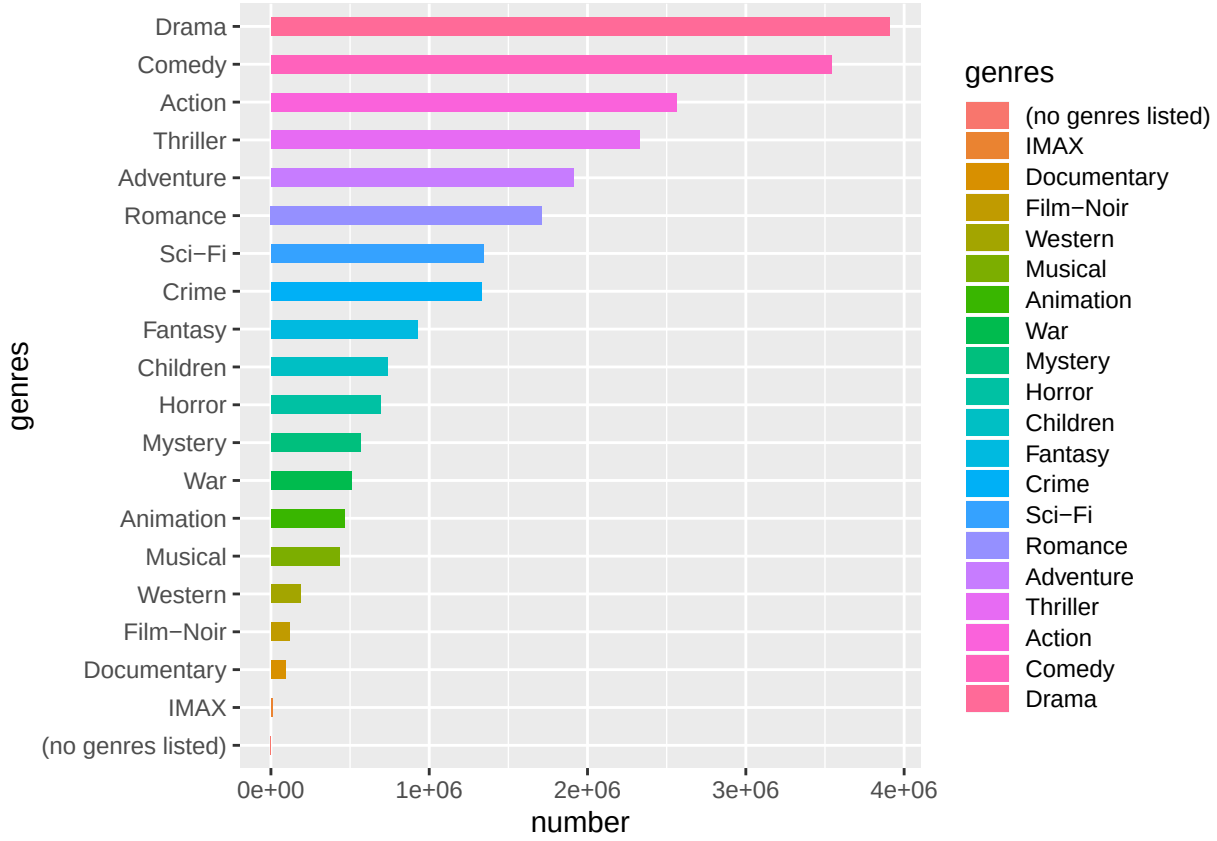
```
movies_by_rating <- edx %>% separate_rows(genres, sep = "\\|") %>%
                            group_by(genres) %>%
                            summarise(number = n()) %>%
                            arrange(desc(number))
movies_by_rating
```

```
## # A tibble: 20 x 2
##    genres              number
##    <chr>                <int>
##  1 Drama              3910127
##  2 Comedy             3540930
##  3 Action             2560545
##  4 Thriller           2325899
##  5 Adventure          1908892
##  6 Romance            1712100
##  7 Sci-Fi             1341183
##  8 Crime              1327715
```

```
##  9 Fantasy             925637
## 10 Children            737994
## 11 Horror              691485
## 12 Mystery             568332
## 13 War                 511147
## 14 Animation           467168
## 15 Musical             433080
## 16 Western             189394
## 17 Film-Noir           118541
## 18 Documentary          93066
## 19 IMAX                  8181
## 20 (no genres listed)       7
```

Bar plotting the results above.

```
movies_by_rating %>%
  mutate(genres = reorder(genres, number)) %>%
  ggplot(aes(genres, number, fill = genres)) +
  geom_bar(width = 0.5, stat = "identity") +
  coord_flip() +
  theme(legend.key.height = unit(0.8,"line"))
```



As we can see, drama movies have the most ratings among all the genres, followed by comedy and action in the second and third place, respectively. Moreover, small percentages of the genres are labeled as "IMAX" and "no genres listed", which definitely is a mistake. Nervertheles, we can neglect such contributions since the total occurrence is very low [8188 in total (~0.035%)].

Word cloud of the results:

```
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
movies_by_rating_2 <- edx %>%
  separate_rows(genres, sep = "\\|") %>%
  group_by(genres) %>%
  summarize(Ratings_perGenre_Sum = n(),
            Ratings_perGenre_Mean = mean(rating),
            Movies_perGenre_Sum = n_distinct(movieId),
            Users_perGenre_Sum = n_distinct(userId))

wordcloud(words = movies_by_rating_2$genres, freq = movies_by_rating_2$Ratings_perGenre_Sum,
    min.freq = 10, max.words = 10, random.order = FALSE, random.color = FALSE,
    rot.per = 0.35, scale = c(5, 0.2), font = 4, colors = brewer.pal(8, "Dark2"),
    main = "Most rated genres")
```


```
percentage.mistake.genres <- (8188/sum(movies_by_rating$number))*100
cat("Percentage of contributions that are misclassified:", percentage.mistake.genres, "%")
```

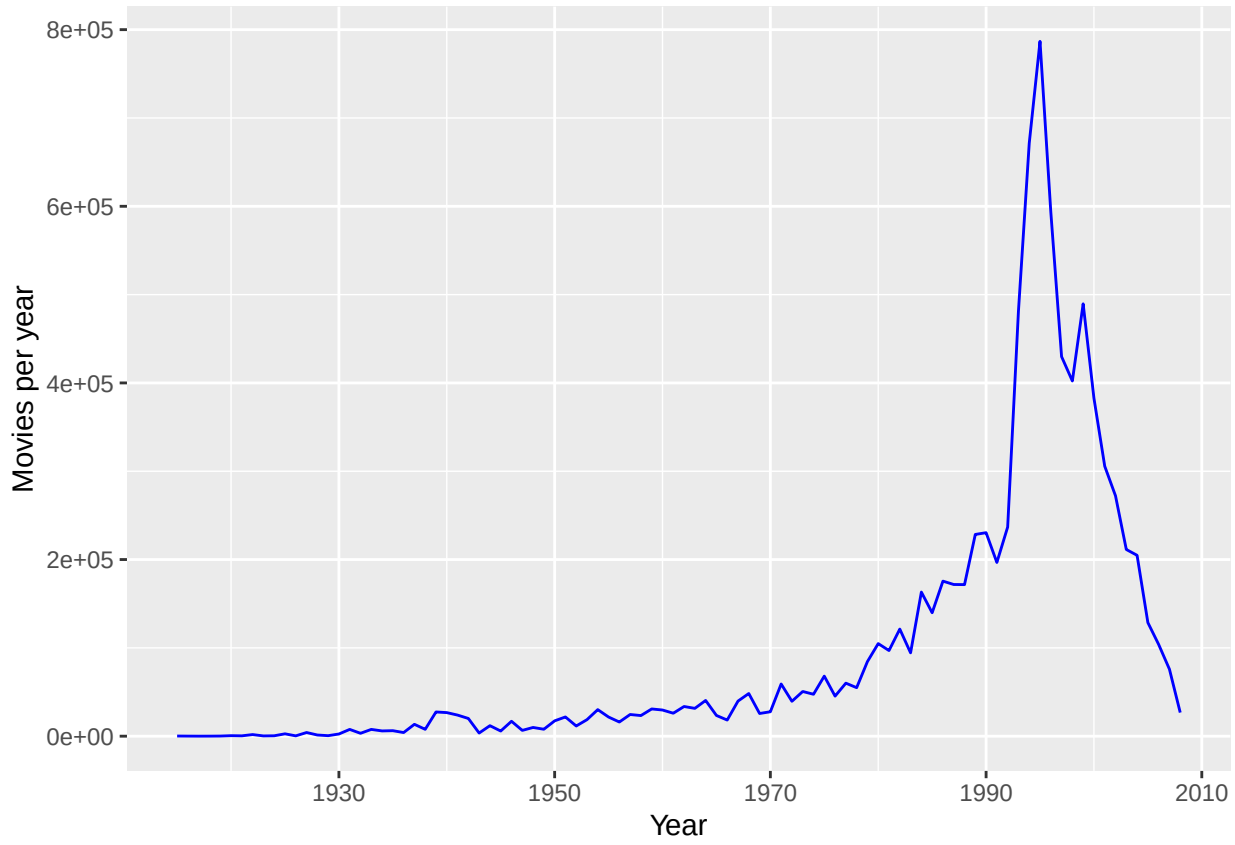
```
## Percentage of contributions that are misclassified: 0.03503424 %
```

##Movies per year Now, let's jus see how the number of movies (per genre) has evolved over the years.

```
#library(lubridate)
movies_year <- edx %>%
  extract(title, c("title", "year"), regex = "^(.*) \\(([0-9 \\-]*)\\)$", remove = F)

movies_per_year <- movies_year %>%
  select(movieId, year) %>% # select columns we need
  group_by(year) %>% # group by year
  summarise(count = n())  %>% # count movies per year
  arrange(year)

head(movies_per_year,10)
```

```
## # A tibble: 10 x 2
##    year  count
##    <chr> <int>
##  1 1915    180
##  2 1916     84
##  3 1917     32
##  4 1918     73
##  5 1919    158
##  6 1920    575
##  7 1921    406
##  8 1922   1825
##  9 1923    316
## 10 1924    457
```

```
movies_per_year_df <- as.data.frame(movies_per_year)

movies_per_year_df %>%
  ggplot(aes(x = as.numeric(year), y = count)) +
  geom_line(color="blue") +
  xlab("Year") +
  ylab("Movies per year")
```




```
movies_per_year$year[which.max(movies_per_year$count)]
```

```
## [1] "1995"
```

Here we see that the year of 1995 was the peak for movie productions.

#Classification models ## Model 1: Average rating In this first model, we will assume that the predictions are equal to the mean rating values and just below the mean. With these simple assumptions, let's evaluate the root mean squared errors (RMSE) for both approaches.

```
#Splitting the edx dataset into train and test
split_index <- createDataPartition(y = edx$rating, times = 1, p = 0.2, list = FALSE)
edx_train <- edx[-split_index,]
edx_test <- edx[split_index,]

# Root mean squared error
RMSE <- function(true_ratings, predicted_ratings){
     sqrt(mean((true_ratings - predicted_ratings)^2))
}

### Building the Recommendation System

mu_hat <- mean(edx_train$rating)
mu_hat
```

```
## [1] 3.512493
```

```
#Prediction using meand rating value
naive_rmse <- RMSE(edx_test$rating, mu_hat)
naive_rmse
```

```
## [1] 1.060233
```

```
#Prediction using only values of 2.5 as rating
predictions <- rep(2.5, nrow(edx_test))
less_than_mean <- RMSE(edx_test$rating, predictions)

#Table of RMSE results
rmse_results <- tibble(method = c("Just the average", "Lower than average (2.5)"), RMSE = c(naive_rmse,
rmse_results
```

```
## # A tibble: 2 x 2
##   method                    RMSE
##   <chr>                    <dbl>
## 1 Just the average          1.06
## 2 Lower than average (2.5)  1.47
```

As we can note, both approaches are quite inefficient and performs really poorly. However, they will be used to compare the other approaches, therefore we expect that the further solutions should perform better than random guesses.

Model 2: Item-based and user-based collabirative systems

In one case, we are going to create an Item-Item matrix (ignoring the user), focusing on what items from all the options are more similar to what we know the user enjoys. On the other case, a User-Item Matrix will be created, which will predict the ratings on items the active user has not see, based on the other similar users.

```
#Getting the mean ratings value
mu <- mean(edx$rating)

movie_avgs <- edx %>%
     group_by(movieId) %>%
     summarize(b_i = mean(rating - mu))

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

model_1_rmse <- RMSE(validation$rating, predicted_ratings_movies)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie Effect Model",
                                 RMSE = model_1_rmse ))
#rmse_results %>% knitr::kable()

user_avgs <- edx %>%
     left_join(movie_avgs, by='movieId') %>%
     group_by(userId) %>%
```

```
     summarize(b_u = mean(rating - mu - b_i))

predicted_ratings_movies_users <- validation %>%
     left_join(movie_avgs, by='movieId') %>%
     left_join(user_avgs, by='userId') %>%
     mutate(pred = mu + b_i + b_u) %>%
     .$pred

model_2_rmse <- RMSE(validation$rating, predicted_ratings_movies_users)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="Movie + User Effects Model",
                                     RMSE = model_2_rmse ))
#rmse_results %>% knitr::kable()
rmse_results
```

```
## # A tibble: 4 x 2
##   method                      RMSE
##   <chr>                      <dbl>
## 1 Just the average           1.06
## 2 Lower than average (2.5)   1.47
## 3 Movie Effect Model         0.944
## 4 Movie + User Effects Model 0.865
```

As we can observe, both models perform better than simple guesses of the first ones. The model where we take into account both item and user matrix has the minimum RMSE value, meaning that this model is a better reccomendation engine so far.

Model 3: Regularization

Regularization is a technique to cope with over-fitting which comes up in training a model on sample data. With the regularization we are able to smooth out the noise, wich can lead to larger errors. In this case, we are going to consider both item and user-based regularization to see if we can perform better the models used before.

```
mu <- mean(edx$rating)
lambdas <- seq(0, 10, 0.25)

just_the_sum <- edx %>%
     group_by(movieId) %>%
     summarize(s = sum(rating - mu), n_i = n())

rmses_movies <- sapply(lambdas, function(l){
     predicted_ratings_reg1 <- validation %>%
          left_join(just_the_sum, by='movieId') %>%
          mutate(b_i = s/(n_i+l)) %>%
          mutate(pred = mu + b_i) %>%
          .$pred
     return(RMSE(validation$rating, predicted_ratings_reg1))
})

rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularized Movie Effect Model",
```

```
                                    RMSE = min(rmses_movies)))
lambdas <- seq(0, 10, 0.25)
rmses_movies_user <- sapply(lambdas, function(l){
     mu <- mean(edx$rating)
     b_i <- edx %>%
          group_by(movieId) %>%
          summarize(b_i = sum(rating - mu)/(n()+l))
     b_u <- edx %>%
          left_join(b_i, by="movieId") %>%
          group_by(userId) %>%
          summarize(b_u = sum(rating - b_i - mu)/(n()+l))
     predicted_ratings_reg2 <-
          validation %>%
          left_join(b_i, by = "movieId") %>%
          left_join(b_u, by = "userId") %>%
          mutate(pred = mu + b_i + b_u) %>%
          .$pred
     return(RMSE(validation$rating, predicted_ratings_reg2))
})

rmse_results <- bind_rows(rmse_results,
                          tibble(method="Regularized Movie + User Effect Model",
                                    RMSE = min(rmses_movies_user)))

rmse_results
```

```
## # A tibble: 6 x 2
##   method                                RMSE
##   <chr>                                <dbl>
## 1 Just the average                     1.06
## 2 Lower than average (2.5)             1.47
## 3 Movie Effect Model                   0.944
## 4 Movie + User Effects Model           0.865
## 5 Regularized Movie Effect Model       0.944
## 6 Regularized Movie + User Effect Model 0.865
```

Altough the regularization model are strong against noises from the dataset, which supposes to be a better pattern finding algorythm by penalizing larger estimates, it is indeed outperformed by the item-user collaborative model.

Model 4: Matrix factorization

Matrix factorization is a class of collaborative filtering algorithm that works by decomposing the user-item interaction matrix into the product of two lower dimensionality rectangular matrices. For this purpose, we are going to use the recosystem library, which is a implementation of matrix factorization using R as a wrapper of LIBMF, a high-performance C++ library.

```
library(recosystem)
train_data <- data_memory(user_index = edx_train$userId,
                          item_index = edx_train$movieId,
                          rating = edx_train$rating,
                          index1 = T)
```

```
test_data <- data_memory(user_index = edx_test$userId,
                         item_index = edx_test$movieId,
                         rating = edx_test$rating,
                         index1 = T)
rec <- Reco()
rec$train(train_data,
          opts = c(dim = 30,
                   costp_l2 = 0.1,
                   costq_l2 = 0.1,
                   lrate = 0.1,
                   niter = 100,
                   nthread = 4,
                   verbose = F)
          )

predictions_mf <- rec$predict(test_data, out_memory())

model_mf_rmse <- RMSE(edx_test$rating, predictions_mf)
rmse_results <- bind_rows(rmse_results,
                          data_frame(method="Matrix factorizartion Model",
                                     RMSE = model_mf_rmse
                                     )
                          )
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

```
# Arraging in descending order
rmse_results %>% arrange(desc(RMSE))
```

```
## # A tibble: 7 x 2
##   method                                RMSE
##   <chr>                                <dbl>
## 1 Lower than average (2.5)             1.47
## 2 Just the average                     1.06
## 3 Movie Effect Model                   0.944
## 4 Regularized Movie Effect Model       0.944
## 5 Movie + User Effects Model           0.865
## 6 Regularized Movie + User Effect Model 0.865
## 7 Matrix factorizartion Model          0.812
```

The RMSE values for the matrix factorization shows that this model outperforms every other one proved so far. This model has proven to be very powerful and, in this scenario, is the best algorythm to be used as a recommendation engine.

Conclusions

To sum up, after exploring, visualizing and modelling the 10 million movie ratings dataset, we were able to determine the preferred genres, the best rated movies and, also, to create an algorythm to be used as a recommendation engine. In this aspect, the matrix factorization model is the best one, which give us a RMSE of approximately 0.81.