Capstone Project: Movielens Recommender System

Matej Salamunić

02/11/2020

1 Introduction

This project aims to create a movie recommendation system using the MovieLens dat aset. The complete MovieLens data set contains 27 million ratings on 58,000 movies by 280,000 users (https://grouplens.org/datasets/movielens/latest/). To make computation easier this project used 10 million subset of MovieLens data set with 10 million ratings on 10,000 movies by 72,000 users (https://grouplens.org/datasets/movielens/10m/). The 10M MovieLens is downloaded and then data is explored to understand the features and the structure of the data set. After data exploration and visualization in data preparation step movie ID, user ID, movie title, rating and released year are taken for the model analysis.

To develop movie recommendation system algorithms 10M MovieLens data set is randomly divided into a set called "edx" (90%), and a "validation" set (10%). The "edx" set is split again into two subsets used for training ("train_edx", 80%) and testing ("test_edx", 20%). Models are first trained on "train_edx" and tested on "test_edx" data set. Linear model with movie, user and release year effect is evaluated first, then regularization model that penalize samples with few ratings. Finally, the matrix factorization model based on recosystem package is evaluated on the user - movie matrix. To compare the performance of different algorithms Root Mean Square Error (RMSE) was used.

For two systems with lowest RMSE value, regularization model and matrix factorization model, final evaluation is performed against the "validation" set. At the end rating prediction is performed for the project best recommendation system based on the matrix factorization model.

Information on the available scripts, implementation system and processing time can be found in the appendix.

2 Method and Analysis

2.1 Download Data and Generate Data Sets

This section describes the required packages in the project, the data source used and the data sets created from the sources with which the evaluation of different algorithms was performed.

2.1.1 Install packages and load library

First it is necessary to download and install the R packages used in this project (tidyverse, lubridate, caret, data.table, dslabs, ggthemes, scales, recosystem, knitr, kableExtra, ggplot2, gridExtra, jjb, naniar). After installation, to be used they have to be loaded into the session.

2.1.2 Download Data

The data source for the project is a 10M version of the MovieLens data sets (http://files.grouplens.org/datasets/movielens/ml-10m.zip) generated by the GroupLens research lab. The 10M MovieLens files are downloaded so that can be used to create the data sets used in the project.

2.1.3 Generate Data Sets

Split the downloaded MovieLens data set into **edx** set and **validation** set. The edx set will be 90% and validation set will be 10% of MovieLens data set. Then, split the edx set in two parts - **train_edx** with 80% and **test_edx** with 20% of edx set data.

2.2 Exploration and Vizualization

Insight into the basic characteristics and properties of the data is obtained by inspecting the edx data set.

Table 1: edx

userId	movieId	rating	timestamp	title	genres
1	122	5	838985046	Boomerang (1992)	Comedy|Romance
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi

```
str(edx)

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

Initial result show that edx has 9,000,055 rows for 6 columns (userId, movieId, rating, timestamp, title, geners). In the edx data set each row has one observation and the column names are the features name. The users information are stored in userId column. The movie information is in movieId and title columns, and there is a movie release year at the end of the title string. The rating date is available in timestamp and each movie has one or more genre in the genres column.

2.2.1 About Rating

To see how often a certain rank is used, which are used the most and which the least, we will calculate the total number for each rank in the edx data set.

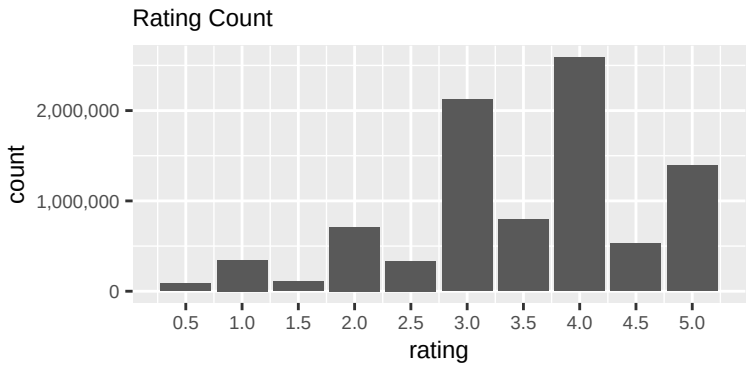


Table 2: Rating count

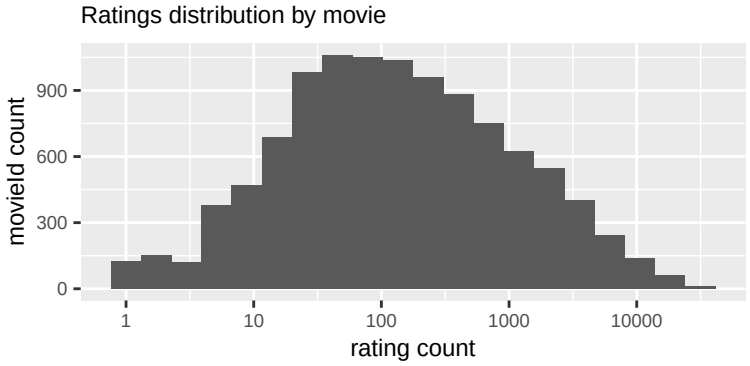
rating	0.5	1	1.5	2	2.5	3	3.5	4	4.5	5
n	85,374.0	345,679	106,426.0	711,422	333,010.0	2,121,240	791,624.0	2,588,430	526,736.0	1,390,114

From the graph above, it can be seen that users mostly use average ratings (3 stars and 4 stars) and all half stars ranks are less frequent than full stars.

2.2.2 About MovieId

```
format(n_distinct(edx$movieId), big.mark= ',')
```

[1] "10,677" There are 10,677 different movies in the edx data set and theirs distribution by number of rankings is shown in the graph below.

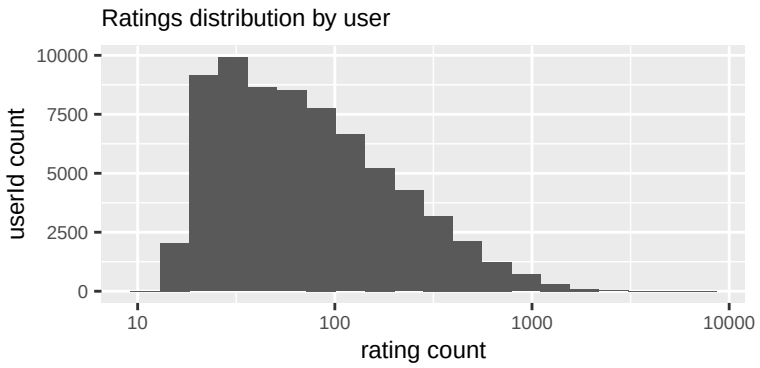


A small number of movies have only one rating or over 1,000 ratings and the majority movies have been rated between 50 and 1,000 times.

2.2.3 About UserId

```
format(n_distinct(edx$userId), big.mark= ',')
```

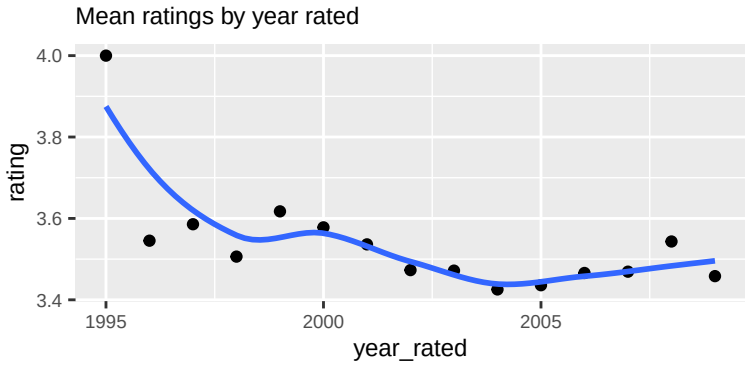
[1] "69,878" There are 69,878 different users in the edx set.



Only few users rated more than a 1,000 times or less than 10 times. Most users rated movies between 30 and 100 times.

2.2.4 About Year Rated (timestamp)

The timestamp variable represents the time and date in which the rating was provided so we can use year of rating from the timestamp. In the figure below we can see the distribution of the average rating according to the year in which the rating was given.



Looking at the year the rating was made there is no significant impact on the average rating, especially in the last 20 years.

2.2.5 About Genres

In the data set movies are associated with several different values for the genre variable. Thus e.g. movie Outbreak with movieId 292 has 4 separated values in the genres variable (Action|Drama|Sci-Fi|Thriller).

```
n_distinct((edx$genres))
```

[1] 797

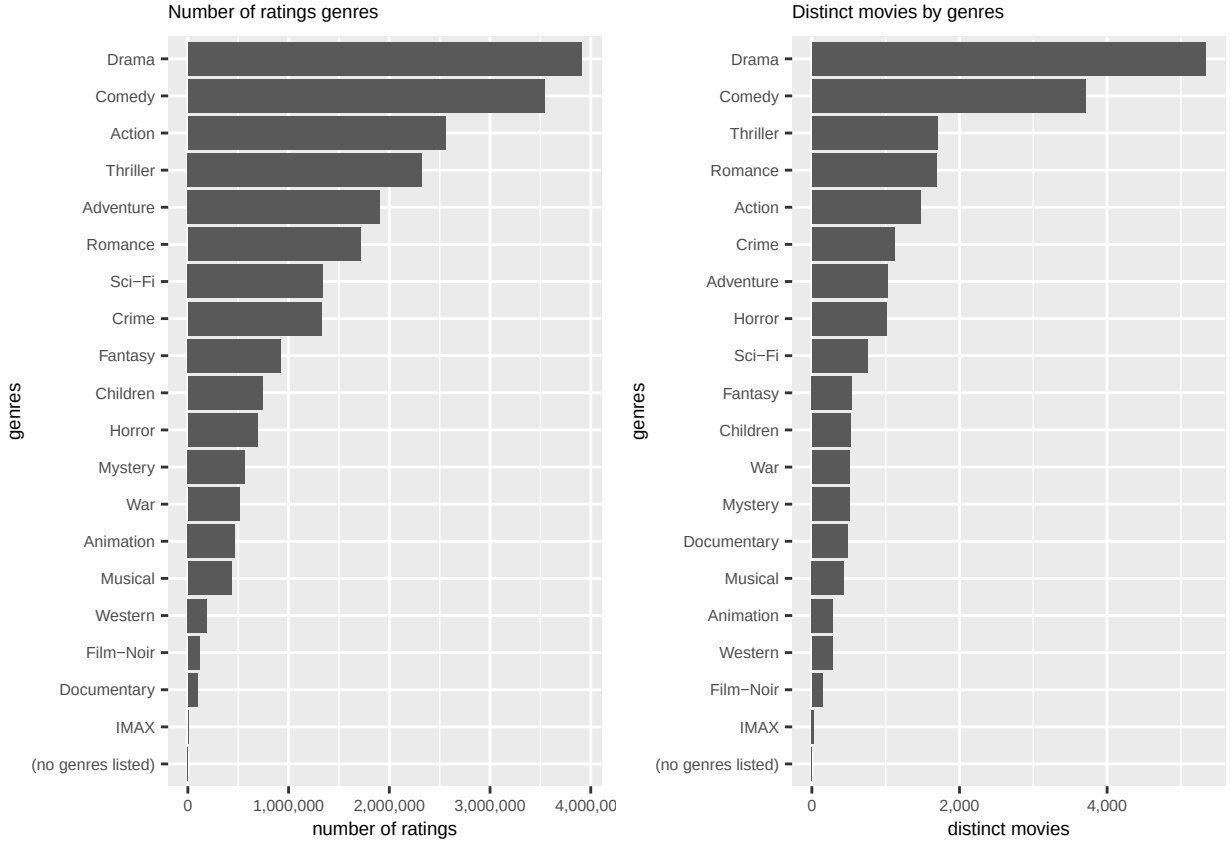
The edx data set has 797 distinct genres. The previously described genre variable in the edx data set has over 700 different values. In order to examine an individual genres characteristic, we will split the genres information into multiple row into the genres_edx data set.

```
genres_edx <- edx %>% separate_rows(genres, sep ="\\|")
```

```
##     data_set       rows
## 1        edx  9,000,055
## 2 genres_edx 23,371,423
```

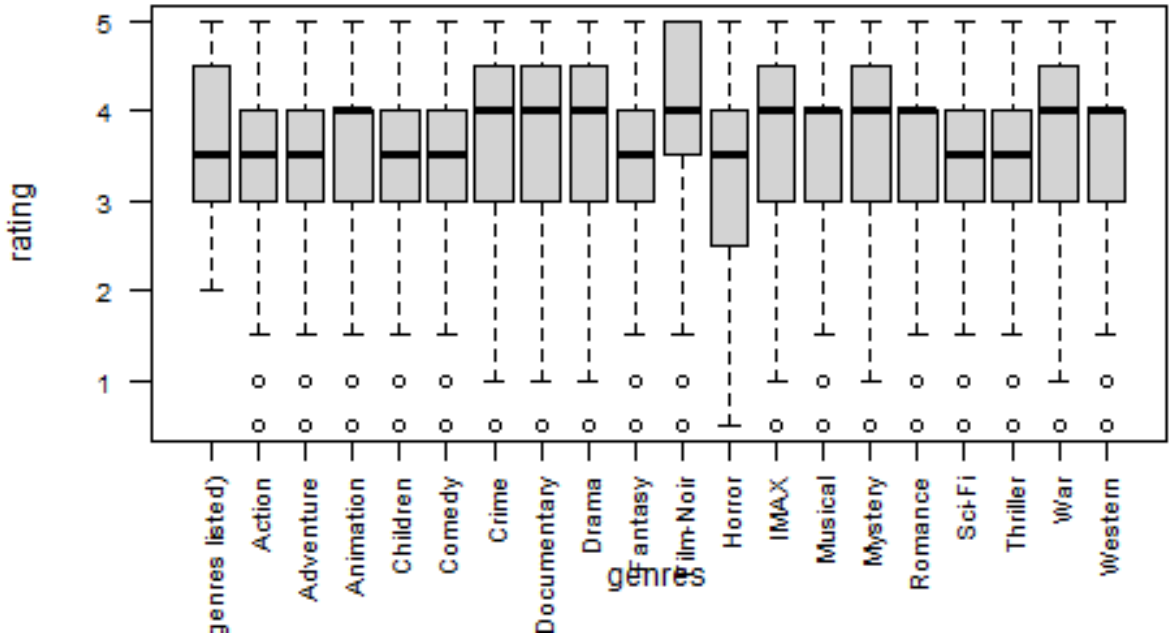
The edx data set has about 9 million rows and when we separate the data in the genre column we get a data set (genres_edx) of 23 million rows. This data transformation duplicates the ratings of an individual movie as many times as there are different genre values and in this data set (genres_edx) this affects the film rating information.

The images below explore the rating numbers and number of movies in each genre from the genres_edx dataset, so one movie can appear in several genres.



Drama and Comedy are the highest rated types of genres, but at the same time these two genres have a larger number of movies than other genres. Therefore we cannot say that one type ranks more often than another.

The following graph shows the average rating per each genres.



Some genres have a higher average rating (Film-Noir, War) while some are below average (Horror, Sci-Fi). But overall it's not a big difference between genres given the average rating per genre.

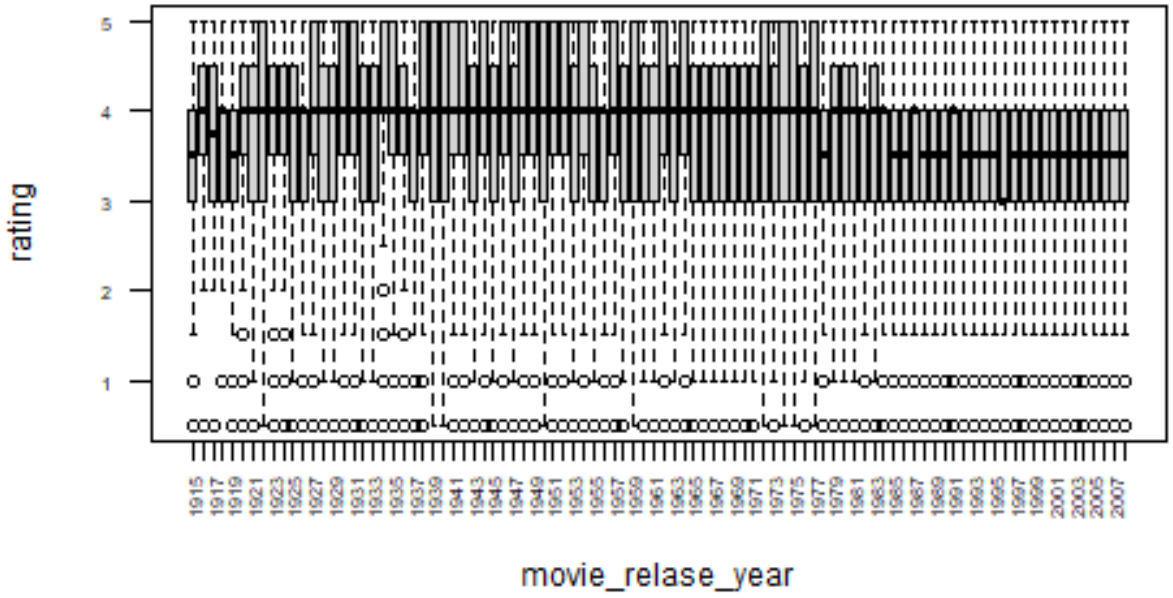
2.2.6 About Movie Relase Year

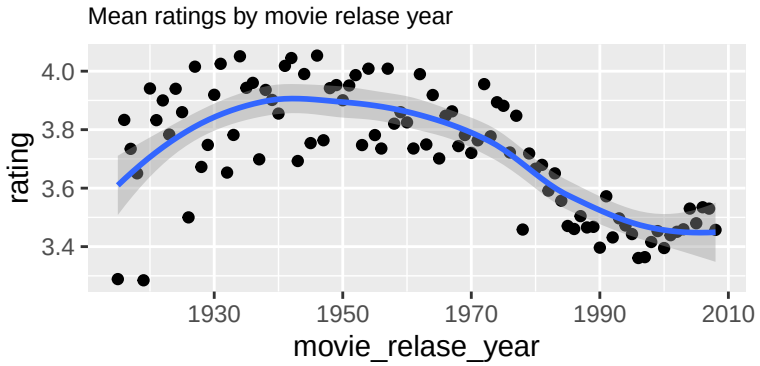
The title column in the edx data set contains the title of the movie and also the movie release year information which can be extracted into a special column (movie_relase_year).

```
#movie release year
movie_relase_year_edx <- edx %>%
  mutate (movie_relase_year=as.numeric(str_sub(title, -5, -2)))
```

userId	movieId	rating	timestamp	title	genres	movie_relase_year
1	122	5	838985046	Boomerang (1992)	Comedy|Romance	1992
1	185	5	838983525	Net, The (1995)	Action|Crime|Thriller	1995
1	292	5	838983421	Outbreak (1995)	Action|Drama|Sci-Fi|Thriller	1995
1	316	5	838983392	Stargate (1994)	Action|Adventure|Sci-Fi	1994
1	329	5	838983392	Star Trek: Generations (1994)	Action|Adventure|Drama|Sci-Fi	1994

The figures below show the distribution of the average rating of movies according to the year in which the movie was released.





Given the year the film was released, it can been seen that movies in recent years, since 1995, have received on average a lower rating than older films.

2.2.7 User - Movie Matrix

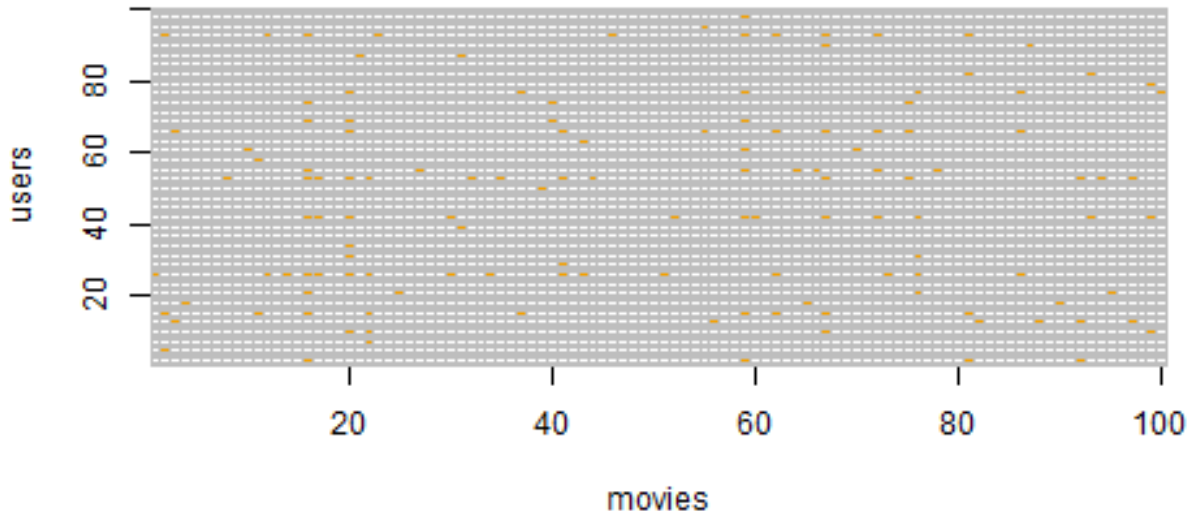
Users only rate the movies they want to rate, so not every movie is rated by the all users and that can be seen in the user - movie matrix.

```
#user - movie rating matrix
matrix_all_edx <- edx %>%
  select(userId, movieId, rating) %>%
  mutate(rating = 1) %>%
  spread(movieId, rating) %>% as.matrix()
```

The table below shows how much empty data there is in the matrix (matrix_all_edx).

```
##     data        rows
## 1    all 746,157,284
## 2  empty 737,087,351
## 3 filled   9,069,933
```

The figure below shows for 100 randomly selected users and movies which movie is rated by which user (filled fields) and unrated films are those with empty fields.



The edx data set has 10,677 different movies, 69,878 different users and has about 9 million lines (rating). If each user ranked each movie, the edx data set would have more than 700 million rows. Thus, it follows that only less than 2% of all possible rankings are present, and the user - movie matrix is about 98% sparse.

2.3 Preprocessing, Data Cleaning and Prepare

The data exploration and visualization showed that some movies were rated more than others, and that some users ranked more often than others. Looking at the movie release year, it was noticed that newer movies have a slightly lower rating on average. Movie genres and the year when the movie is ranked do not significantly affect the average rating. Also, it has been observed that movies, regardless of the release year or the genre of the movie, are rarely ranked with ratings lower than 1.5, and that the half star ratings are less common than whole star ratings. If we transform an edx data set into a user - movie matrix a large amount of cells remain empty.

Based on the data exploration movieId, userId, year of relase are information that will be used for the machine learning algorithms evaluation, so timestamp and generes can be excluded form data sets.

```
#preparing data sets
edx <- edx %>% select(userId, movieId, rating, title) %>%
               mutate (movie_relase_year=as.numeric(str_sub(title, -5, -2)))

train_edx <- train_edx %>% select(userId, movieId, rating, title) %>%
               mutate (movie_relase_year=as.numeric(str_sub(title, -5, -2)))
```

```
test_edx  <- test_edx  %>% select(userId, movieId, rating, title) %>%
               mutate (movie_relase_year=as.numeric(str_sub(title, -5, -2)))

validation <- validation %>% select(userId, movieId, rating, title) %>%
               mutate (movie_relase_year=as.numeric(str_sub(title, -5, -2)))
```

Table 3: modeling information

userId	movieId	rating	title	movie_relase_year
1	122	5	Boomerang (1992)	1992
1	185	5	Net, The (1995)	1995
1	292	5	Outbreak (1995)	1995
1	316	5	Stargate (1994)	1994
1	329	5	Star Trek: Generations (1994)	1994

Table 4: Data sets

MovieLens_split	rows	edx_split	rows
edx	9,000,055	train_edx	7,200,089
validation	999,999	test_edx	1,799,966

2.4 Modeling Approach

First, a basic model was made which is just the mean of the observed values. The following three models add a movie effect, a user effect, and a movie release year effect. The fifth and sixth model uses regularization parameter that penalizes movies with few ratings. The fifth model uses a user and movie effect with regularization parameter and sixth model uses a user, movie and movie release year effect with regularization parameter. The seventh model uses userId, movieId and ratings data for the matrix factorization model from recosystem package that use cross validation to tune the model parameters.

Each model is trained on "train_edx" and then tested on "test_edx" data and finally the resulting Root Mean Square Error (RMSE) value of the model is calculated.

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

Assessments of these models is based on their resulting Root Mean Square Error (RMSE). The RMSE value was calculated for each model and listed in the scoreboard (RMSE results) and those with lower RMSE were better.

2.4.1 Model 1 - Baseline

Model 1 is based only on ratings data and predicts that all userId will give the same rating to all movieId. The average minimizes the RMSE so prediction rating for this model is the average of all observed ratings.

```
#calculating the mean rating value
m1_mean <- mean(train_edx$rating)
m1_mean
```

```
## [1] 3.512478
```

The calculated mean rating (m1_mean) is the baseline prediction, and the resulting RMSE calculated on the test_edx data set is as the baseline RMSE (1 Baseline in the table below).

```
#result for Model 1 - Baseline
m1_rmse <- RMSE(test_edx$rating, m1_mean)
m1_rmse
```

```
## [1] 1.059904
```

Table 5: RMSE results

Model	RMSE
1 Baseline	1.059904

The following models trying to achieve the lowest possible RMSE compared to the baseline.

2.4.2 Model 2 - Movie Effect

The movie itself affects the rating, and different movies are rated differently because some movies are more popular than others and users have different preferences. This is movie effect and it means that the average rating on specific movie has a difference from the overall average rating of all movies.
Train Model 2 for movie effect on train_edx data set:

```
#train model for movie effect
m2_avgs_movie <-
  train_edx %>%
  group_by(movieId) %>%
  summarize(avgs_movie = mean(rating - m1_mean), .groups = "drop")
```

Test Model 2 with movie effect on test_edx data set,

```
#test movie effect model
pred_m2_avgs_movie <- m1_mean + test_edx   %>%
  left_join(m2_avgs_movie, by = "movieId") %>%
  .$avgs_movie
```

and calculate RMSE:

```
#RMSE calculation and listing in the results table
rmse_table <- bind_rows(rmse_table, tibble(Model = "2 Movie Effect",
                        RMSE=RMSE(test_edx$rating, pred_m2_avgs_movie)))
```

Table 6: RMSE results

Model	RMSE
1 Baseline	1.0599043
2 Movie Effect	0.9437429

By adding a movie effect, the RMSE is reduced to a value lower than the baseline.

2.4.3 Model 3 – Movie and User Effects

Different users have different ways of rating. Some users give a high rating to each movie, while others always give a low rating. In Model 3, we use this user effect by adding it to the movie effect from the previous model since both affect rating prediction.

Train Model 3 for movie and user effects on train_edx data set:

```
#train model for movie and user effects
m3_avgs_user <-
  train_edx %>%
  left_join(m2_avgs_movie, by = "movieId") %>%
  group_by(userId) %>%
  summarize(avgs_user = mean(rating - m1_mean - avgs_movie), .groups = "drop")
```

Test Model 3 with movie and user effects on test_edx data set and calculate RMSE:

```
#test movie and user effects model
pred_m3_avgs_user <- test_edx %>%
  left_join(m2_avgs_movie, by = "movieId") %>%
  left_join(m3_avgs_user, by = "userId") %>%
  mutate(pred_m3 = m1_mean + avgs_movie + avgs_user) %>%
  .$pred_m3
```

```
#RMSE calculation and listing in the results table
rmse_table <- bind_rows(rmse_table, tibble(Model = "3 Movie and User Effects",
                        RMSE=RMSE(test_edx$rating, pred_m3_avgs_user)))
```

Table 7: RMSE results

Model	RMSE
1 Baseline	1.0599043
2 Movie Effect	0.9437429
3 Movie and User Effects	0.8659319

Adding a user effect to the movie effect considerably improves RMSE compared to the previous model with only a movie effect.

2.4.4 Model 4 – Movie, User and Release Year Effects

From the exploration part, it turns out that the movie release year has effect on ratings, so in Model 4, the effect of the year of movie release is added to the effects of the movie and user.

Train Model 4 for movie, user and release year effects on train_edx data set:

```
#train model for movie, user and release year effects
m4_avgs_relase <-
  train_edx %>%
  left_join(m2_avgs_movie, by = "movieId") %>%
  left_join(m3_avgs_user, by = "userId") %>%
  group_by(movie_relase_year) %>%
  summarize(avgs_relase = mean(rating - m1_mean - avgs_movie - avgs_user), .groups = "drop")
```

Test Model 4 with movie, user and release year effects on test_edx data set and calculate RMSE:

```
#test movie, user and release year effects model
pred_m4_avgs_relase <- test_edx %>%
  left_join(m2_avgs_movie, by = "movieId") %>%
  left_join(m3_avgs_user, by = "userId") %>%
  left_join(m4_avgs_relase, by = "movie_relase_year") %>%
  mutate(pred_m4 = m1_mean + avgs_movie + avgs_user+avgs_relase) %>%
  .$pred_m4
```

```
#RMSE calculation and listing in the results table
rmse_table <- bind_rows(rmse_table, tibble(Model = "4 Movie, User and Release Year Effects",
                        RMSE=RMSE(test_edx$rating,pred_m4_avgs_relase)))
```

Table 8: RMSE results

Model	RMSE
1 Baseline	1.0599043
2 Movie Effect	0.9437429
3 Movie and User Effects	0.8659319
4 Movie, User and Release Year Effects	0.8656117

From the results table above it can be seen that adding the movie release year effect only slightly reduced the RMSE compared to the previous model.

2.4.5 Model 5 – Regularized Movie and User Effects

In the data exploration part it is observe that some movies are rated very few times, and there are users who have rated only few times. Sample size for these users and movies is small and this can lead to large estimated error and get worse the RMSE result. In this Model 5, the estimated value is regulated by a penalty factor that is added to penalizes small sample sizes and otherwise has little or no impact.

Next function regularize user and movie effects adding a penalty factor lambda and return RMSE result. This function is used to find optimal lambda value that minimizes the RMSE.

```
#regularization function with lambda parameter (movie and user effects)
regularization_m_u2 <- function(lambda, train_set, test_set){

  #baseline model
  m1_mean <- mean(train_set$rating)

  #train movie effect regularized with lambda
  m2_avgs_movie <-
    train_set %>%
    group_by(movieId) %>%
    summarize(avgs_movie = sum(rating - m1_mean)/(n()+lambda), .groups = "drop")

  #train user effect regularized with lambda
  m3_avgs_user <-
    train_set %>%
    left_join(m2_avgs_movie, by = 'movieId') %>%
    group_by(userId) %>%
```

```
    summarize(avgs_user = sum(rating - m1_mean - avgs_movie)/(n()+lambda), .groups = "drop")

  #test regularized movie and user effects model
  pred_m3_avgs_user <-
    test_set %>%
    left_join(m2_avgs_movie, by = "movieId") %>%
    left_join(m3_avgs_user, by = "userId") %>%
    mutate(pred_m3 = m1_mean + avgs_movie + avgs_user) %>%
    .$pred_m3

  #return RMSE for regularized model with lambda parameter
  return(RMSE(pred_m3_avgs_user, test_set$rating))
}
```

First, a set of lambda values is defined, and then the RMSE for each lambda is calculated. Finally, the best lambda resulting in the lowest RMSE for this model was selected to perform the prediction.

```
#define a set of lambdas
lambda_set <- seq(0, 10, 0.25)

#calculate RMSE for each lambda
rmses_L <- sapply(lambda_set, regularization_m_u2,
                  train_set=train_edx, test_set=test_edx)

#take lambda which returns the lowest RMSE, best lambda
lambda <- lambda_set[which.min(rmses_L)]
```

Train Model 5 on train_edx data set to calculate the predicted rating using the best lambda parameters achieved from regularization (lambda which return lowest RMSE):

```
#train Model 5 using best lambda parameter
#train base lane model
m1_mean <- mean(train_edx$rating)

#train regularized movie effect model
m2_avgs_movie <-
  train_edx %>%
  group_by(movieId) %>%
  summarize(avgs_movie = sum(rating - m1_mean)/(n()+lambda), .groups = "drop")

#train regularized user effect model
m3_avgs_user <-
  train_edx %>%
  left_join(m2_avgs_movie, by = "movieId") %>%
  group_by(userId) %>%
  summarize(avgs_user = sum(rating - m1_mean - avgs_movie)/(n()+lambda), .groups = "drop")
```

Test Model 5 with regularized movie and user effects on test_edx data set and calculate RMSE:

```
#test regularized model
pred_reg_m3_avgs_user <- test_edx %>%
  left_join(m2_avgs_movie, by = "movieId") %>%
```

```
  left_join(m3_avgs_user, by = "userId") %>%
  mutate(pred_m3 = m1_mean + avgs_movie + avgs_user) %>%
  .$pred_m3
```

```
#RMSE calculation and listing in the results table
rmse_table <- bind_rows(rmse_table, tibble(Model = "5 Regularized Movie and User Effects",
                          RMSE=RMSE(test_edx$rating, pred_reg_m3_avgs_user)))
```

Table 9: RMSE results

Model	RMSE
1 Baseline	1.0599043
2 Movie Effect	0.9437429
3 Movie and User Effects	0.8659319
4 Movie, User and Release Year Effects	0.8656117
5 Regularized Movie and User Effects	0.8652421

The result of Model 5, with regularization of movie and user effects, shows little improvement compared to the model without regularization (Model 3).

2.4.6 Model 6 – Regularized Movie, User and Release Year Effects

In the same way as in the previous model, in Model 6 regularization is applied to movie, user and release year to penalizes small sample sizes.

First a function which is similar to that in the previous model, is defined to regularize user, movie and release year effects adding a penalty factor lambda and return RMSE result.

This function is used to find optimal lambda value that minimizes the RMSE. A set of lambda values is defined, and the RMSE for each lambda is calculated.

The best lambda, resulting in the lowest RMSE for this model, was selected to perform the prediction.

```
#take lambda which returns the lowest RMSE, best lambda
lambda <- lambda_set[which.min(rmses_L)]
lambda
```

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

Than Model 6 is trained on train_edx data set to calculate the predicted rating using the best lambda which return lowest RMSE parameters achieved from regularization on movie, user and release year effects:

```
#train model using best lambda parameter

#train base lane model
m1_mean <- mean(train_edx$rating)

#train regularized movie effect model
m2_avgs_movie <-
  train_edx %>%
  group_by(movieId) %>%
  summarize(avgs_movie = sum(rating - m1_mean)/(n()+lambda), .groups = "drop")
```

```
#train regularized user effect model
m3_avgs_user <-
  train_edx %>%
  left_join(m2_avgs_movie, by = "movieId") %>%
  group_by(userId) %>%
  summarize(avgs_user = sum(rating - m1_mean - avgs_movie)/(n()+lambda), .groups = "drop")

#train regularized release year effect model
m4_avgs_relase <-
  train_edx %>%
  left_join(m2_avgs_movie, by = "movieId") %>%
  left_join(m3_avgs_user, by = "userId") %>%
  group_by(movie_relase_year) %>%
  summarize(avgs_relase = sum(rating - m1_mean - avgs_movie - avgs_user)/(n()+lambda), .groups = "drop")
```

Test Model 6 with regularized movie, user and release year effects on test_edx data set,

```
#test regularized Model 6
pred_reg_m4_avgs_relase <- test_edx %>%
  left_join(m2_avgs_movie, by = "movieId") %>%
  left_join(m3_avgs_user, by = "userId") %>%
  left_join(m4_avgs_relase, by = "movie_relase_year") %>%
  mutate(pred_m4 = m1_mean + avgs_movie + avgs_user+avgs_relase) %>%
  .$pred_m4
```

and calculate RMSE:

```
#RMSE calculation and listing in the results table
rmse_table <- bind_rows(rmse_table, tibble(Model = "6 Regularized Movie, User and Release Effects",
                          RMSE=RMSE(test_edx$rating, pred_reg_m4_avgs_relase)))
```

Table 10: RMSE results

Model	RMSE
1 Baseline	1.0599043
2 Movie Effect	0.9437429
3 Movie and User Effects	0.8659319
4 Movie, User and Release Year Effects	0.8656117
5 Regularized Movie and User Effects	0.8652421
6 Regularized Movie, User and Release Effects	0.8649761

The result of Model 6 with regularization of movie, user, and release year shows little improvement over model without regularization (Model 4).

Regularization has improved model prediction performance in both models (Model 5 and 6) but only slightly.

2.4.7 Model 7 - Matrix Factorization

In the recommendation system, users rarely rate all items therefore the rating matrix is sparse. In this Model 7, matrix factorization is used to find hidden patterns in large, sparse user - movie matrix. Matrix factorization decompose a large user - movie matrix into the product of two smaller dimension matrices to discover movie rating patterns and patterns of users who rate movies.

For matrix factorization the recosystem package is used. The rcosystem (https://cran.r-project.org/web/packages/recosystem/vignettes/introduction.html) provides matrix factorization solution for recommendation system.

To find predicted values with the recosystem we have to covert data into recosystem input format, create model object (Reco), select best parameters, train the model and than compute predicted values.

First, for using train_edx and test_edx data sets in recosystem, they must be organized in 3 columns (user_index, item_index, rating).

```
#train_edx and test_edx convert to recosystem column format
mf_train_edx <-  with(train_edx, data_memory(user_index = userId,
                                             item_index = movieId,
                                             rating     = rating))

mf_test_edx  <-  with(test_edx,  data_memory(user_index = userId,
                                             item_index = movieId,
                                             rating     = rating))
```

Then, create Reco model reference object and call the tune method to select best tuning parameters.

```
#create model object
mf_reco <-  recosystem::Reco()

#set seed for randomized values
#find best tuning parameters
#this can take very long execution time
set.seed(123, sample.kind = "Rounding")
opts_2 <- mf_reco$tune(mf_train_edx,  opts = list(dim = c(10, 20, 30),
                                                  lrate = c(0.1, 0.2),
                                                  costp_l2 = c(0.01, 0.1),
                                                  costq_l2 = c(0.01, 0.1),
                                                  nthread  = 1, niter = 10))
```

To train Model 7, on train_edx data set, with best tuned parameters, train method is used.

```
#train model calling train, with best parameters from tune
mf_reco$train(mf_train_edx, opts = c(opts_2$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9947   1.0064e+07
##    1       0.8783   8.0724e+06
##    2       0.8459   7.4994e+06
##    3       0.8237   7.1398e+06
##    4       0.8072   6.9009e+06
##    5       0.7948   6.7216e+06
##    6       0.7844   6.5861e+06
##    7       0.7753   6.4763e+06
##    8       0.7672   6.3821e+06
##    9       0.7603   6.3074e+06
##   10       0.7541   6.2408e+06
##   11       0.7485   6.1836e+06
##   12       0.7434   6.1346e+06
##   13       0.7390   6.0905e+06
```

```
##   14       0.7348   6.0524e+06
##   15       0.7310   6.0167e+06
##   16       0.7275   5.9859e+06
##   17       0.7244   5.9587e+06
##   18       0.7215   5.9346e+06
##   19       0.7187   5.9112e+06
```

Test Model 7 and calculate predicted values on test_edx data

```
#test Model 7
pred_mf_reco <-  mf_reco$predict(mf_test_edx, out_memory())
head(pred_mf_reco, 5)
```

```
## [1] 4.092782 5.256333 5.506570 4.905925 4.543767
```

and calculate RMSE:

```
#RMSE calculation and listing in the results table
rmse_table <- bind_rows(rmse_table, tibble(Model = "7 Matrix Factorization",
                        RMSE=RMSE(test_edx$rating, pred_mf_reco)))
```

Table 11: RMSE results

Model	RMSE
1 Baseline	1.0599043
2 Movie Effect	0.9437429
3 Movie and User Effects	0.8659319
4 Movie, User and Release Year Effects	0.8656117
5 Regularized Movie and User Effects	0.8652421
6 Regularized Movie, User and Release Effects	0.8649761
7 Matrix Factorization	0.7907250

The result of Model 7 which is based on matrix factorization using the recosysetm package shows substantially improvement compared to all other models.

3 Results

In order to confirm the obtained RMSE results for the two models with the lowest RMSE (Model 6 and Model 7) a final evaluation is conducted over a data set that has not been used so far (validation data set).

3.1 Final Validation for Model 6 – Regularized Movie, User and Release Year Effects

In Model 6 best lambda value that minimizes the RMSE was selected to perform the prediction, and that lambda is used here final Model 6 evaluation.

```
#Model 6 best lambda, lamda_r_m_u_r
lamda_r_m_u_r
```

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

In final validation Model 6 is trained same way like before, using best lambda regularization parameter for movie, user and release year effects but now on edx data set.

```
#final train Model 6 using best lambda parameter on edx data set
#train base lane model on edx data set
edx_mean <- mean(edx$rating)

#train regularized movie effect model on edx data set
m2_avgs_movie <-
  edx %>%
  group_by(movieId) %>%
  summarize(avgs_movie = sum(rating - edx_mean)/(n()+lamda_r_m_u_r), .groups = "drop")

#train regularized user effect model on edx data set
m3_avgs_user <-
  edx %>%
  left_join(m2_avgs_movie, by = 'movieId') %>%
  group_by(userId) %>%
  summarize(avgs_user = sum(rating - edx_mean - avgs_movie)/(n()+lamda_r_m_u_r), .groups = "drop")

#train regularized release year effect model on edx data set
m4_avgs_relase <-
  edx %>%
  left_join(m2_avgs_movie, by = 'movieId') %>%
  left_join(m3_avgs_user, by = "userId") %>%
  group_by(movie_relase_year) %>%
  summarize(avgs_relase = sum(rating - edx_mean - avgs_movie - avgs_user)/(n()+lamda_r_m_u_r), .groups =
```

In the final validation, Model 6 is tested against a validation data set that has not been used in the analysis of this model so far.

```
#test regularized Model 6 on validation data set
final_pred_reg_m4_avgs_relase <- validation %>%
  left_join(m2_avgs_movie, by = "movieId") %>%
  left_join(m3_avgs_user, by = "userId") %>%
  left_join(m4_avgs_relase, by = "movie_relase_year") %>%
  mutate(pred_m4 = edx_mean + avgs_movie + avgs_user + avgs_relase) %>%
  .$pred_m4
```

Calculate RMSE of the final Model 6 evaluation on the edx and validation data set.

```
#RMSE calculation and listing in the results table
rmse_table <- bind_rows(rmse_table, tibble(Model = "8 Final Validation for Model 6 Regularization m+u+r
                        RMSE=RMSE(validation$rating, final_pred_reg_m4_avgs_relase)))
```

Table 12: RMSE results

Model	RMSE
1 Baseline	1.0599043
2 Movie Effect	0.9437429
3 Movie and User Effects	0.8659319
4 Movie, User and Release Year Effects	0.8656117
5 Regularized Movie and User Effects	0.8652421
6 Regularized Movie, User and Release Effects	0.8649761
7 Matrix Factorization	0.7907250
8 Final Validation for Model 6 Regularization m+u+r	0.8645223

The table above, in line 8, shows the final result of the validation of Model 6 (Regularized Movie, User and Release Year Effects) which is quite similar to the previously calculated RMSE for Model 6 (line 6 in the table above) and that confirms the RMSE for the model with regularization on the movie, user and release year effect.

3.2 Final validation for Model 7 - Matrix Factorization

Previously, in Model 7, matrix factorization model based on recosystem package is resulted with best RMSE result. Here, in the final validation for Model 7, same model with same tuning parameter is used, but now, model is train on the complete edx data set and RMSE is calculated in the validation data set.

In recosystem data sets must be organized in 3 columns (user_index, item_index, rating).

```
#edx and validation data set convert to recosystem format
mf_edx <-  with(edx, data_memory(user_index = userId,
                                 item_index = movieId,
                                 rating     = rating))

mf_validation  <-  with(validation,  data_memory(user_index = userId,
                                                 item_index = movieId,
                                                 rating     = rating))
```

Create Reco model reference object and call the tune method to select best tuning parameters.

```
#create model object
mf_reco_final <-  recosystem::Reco()

#set seed for randomized values
#find best tuning parameters
#this  can take very long execution time
set.seed(123, sample.kind = "Rounding")
opts_final <- mf_reco_final$tune(mf_edx,  opts = list(dim = c(10, 20, 30),
                                                      lrate = c(0.1, 0.2),
                                                      costp_l2 = c(0.01, 0.1),
                                                      costq_l2 = c(0.01, 0.1),
                                                      nthread  = 1, niter = 10))
```

In final validation Model 7 is trained with best tuned parameters on edx data set.

```
#train model calling train, with best parameters from tune
mf_reco_final$train(mf_edx, opts = c(opts_final$min, nthread = 1, niter = 20))
```

```
## iter      tr_rmse          obj
##    0       0.9734   1.2043e+07
##    1       0.8722   9.8708e+06
##    2       0.8384   9.1674e+06
##    3       0.8165   8.7404e+06
##    4       0.8014   8.4757e+06
##    5       0.7899   8.2770e+06
##    6       0.7802   8.1260e+06
##    7       0.7721   8.0078e+06
##    8       0.7652   7.9091e+06
##    9       0.7593   7.8318e+06
##   10       0.7540   7.7626e+06
##   11       0.7494   7.7033e+06
##   12       0.7451   7.6537e+06
##   13       0.7414   7.6096e+06
##   14       0.7379   7.5710e+06
##   15       0.7347   7.5356e+06
##   16       0.7318   7.5040e+06
##   17       0.7292   7.4765e+06
##   18       0.7268   7.4525e+06
##   19       0.7245   7.4293e+06
```

In final validation Model 7 is test on validation data.

```
#test Model 7 on validation data
pred_mf_reco_final <-  mf_reco_final$predict(mf_validation, out_memory())
head(pred_mf_reco_final, 5)
```

```
## [1] 4.510787 5.021412 4.823325 3.433930 4.472344
```

Calculate RMSE of the final Model 7 evaluation on the edx and validation data set:

```
#RMSE calculation and listing in the results table
rmse_table <- bind_rows(rmse_table, tibble(Model = "9 Final Validation for Model 7 Matrix Factorization
                        RMSE=RMSE(validation$rating, pred_mf_reco_final)))
```

Table 13: RMSE results

Model	RMSE
1 Baseline	1.0599043
2 Movie Effect	0.9437429
3 Movie and User Effects	0.8659319
4 Movie, User and Release Year Effects	0.8656117
5 Regularized Movie and User Effects	0.8652421
6 Regularized Movie, User and Release Effects	0.8649761
7 Matrix Factorization	0.7907250
8 Final Validation for Model 6 Regularization m+u+r	0.8645223
9 Final Validation for Model 7 Matrix Factorization	0.7828681

The results of final validation for matrix factorization (line 9 in the table above) confirms the RMSE for the Model 7. RMSE results for matrix factorization model are better than the linear models including those with regularization parameters.

3.3 Rating prediction for Final Validation for Model 7 - Matrix Factorization

With the model giving the best result, a recommendation system based on matrix factorization, the rating prediction will be performed on validation data set. The validation set was set aside until the final analysis and here we can use it as a set of userId's for which the movie rating within the set should be predicted. So, we take the validation set as if there is no rating value in it and for the users from that set we predict the rating for the movies that are in that set.

```
#prediction with matrix factorization
mf_prediction <- tibble(userId=validation$userId, movieId=validation$movieId,
                        title = validation$title, rating=validation$rating,
                        mf_pred_rating = pred_mf_reco_final) %>%
                        arrange(-mf_pred_rating)
```

As we have values for the rating in the validation set, we can now look at the prediction results (mf_pred_rating) in relation to the users rating value that are within the validation set (rating). So, for the 5 highest predicted rating values (mf_pred_rating):

```
#top 5, highest predicted rating value
head(mf_prediction, 5)
```

```
## # A tibble: 5 x 5
##   userId movieId title                              rating mf_pred_rating
##    <int>   <dbl> <chr>                               <dbl>          <dbl>
## 1  69672    1471 Boys Life 2 (1997)                      5           5.96
## 2  13027     318 Shawshank Redemption, The (1994)        5           5.88
## 3  21611     527 Schindler's List (1993)                 5           5.84
## 4  65052    2959 Fight Club (1999)                       5           5.84
## 5  31766     318 Shawshank Redemption, The (1994)        5           5.83
```

The task of the recommendation is to predict the rating for the movies and to recommend to the user the ones that are predicted to be the most liked. The user - movie matrix from the edx data set contains data on movies and users and blank fields represent movies that users have not rated. If we look at the matrix, we see that for the user we have information about the movies he has rated. For the movies that this user has not rated, there is rating information from other users. As in the example above, we can predict for a user a rating for unrated movies.

Let's take a user (e.g. userId=35305) from an edx data set and predict the ratings for movies from the edx data set that that user didn't rate. Make a list of movies from the edx data set that was not rated by user 35305.

```
#list of movies from the edx data set that was not rated by userId=35305

#user 35305 rated movies list
edx_35305 <- edx %>% filter(edx$userId==35305)
movieID_35305 <- unique(edx_35305$movieId)
#edx movie list
edx_movieId <- unique(edx$movieId)

#movies which user 35305 did not rate
a_b <- setdiff( unique(edx$movieId), unique(edx_35305$movieId))
predict_movieId <- data.table(movieId=a_b)
#add movie title information
```

```
predict_movieId <- left_join(x = predict_movieId, y = edx)
# Remove duplicate rows
predict_movieId <- predict_movieId %>% distinct(movieId, .keep_all= TRUE)
#set all list with  userId=35305
predict_movieId$userId[predict_movieId$userId >= 1] <- 35305
#set all list with  rating=NA
predict_movieId <- predict_movieId %>%
             replace_with_na(replace = list(rating = c(0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5)))

#list of non rating movies for userId=35305, rating=NA, movieId, title
edx_movieId_set_35305 <- predict_movieId
#edx_movieId_set_35305  convert to recosystem format
mf_edx_movieId_set_35305  <-  with(edx_movieId_set_35305,  data_memory(user_index = userId,
                                                                       item_index = movieId,
                                                                       rating     = rating))
```

Predict the rating for user 35305:

```
#rating prediction for userId
pred_mf_edx_movieId_set_35305 <-  mf_reco_final$predict(mf_edx_movieId_set_35305, out_memory())
#head(pred_mf_edx_movieId_set_35305, 5)
mf_prediction_edx_35305 <- tibble(userId=edx_movieId_set_35305$userId,
                                  movieId=edx_movieId_set_35305$movieId,
                                  title = edx_movieId_set_35305$title,
                                  predicted__rating = pred_mf_edx_movieId_set_35305) %>%
                                  arrange(-predicted__rating )

#top 5 predicted for userId 35305
head(mf_prediction_edx_35305 , 5)
```

```
## # A tibble: 5 x 4
##   userId movieId title                                      predicted__rating
##    <dbl>   <dbl> <chr>                                                  <dbl>
## 1  35305    7140 Legend of Leigh Bowery, The (2002)                      6.19
## 2  35305    8536 Intended, The (2002)                                    5.96
## 3  35305   64275 Blue Light, The (Das Blaue Licht) (1932)                 5.93
## 4  35305     318 Shawshank Redemption, The (1994)                        5.78
## 5  35305    3382 Song of Freedom (1936)                                  5.76
```

The table above shows the 5 movies with the highest predicted rating for the specified user.

4 Conclusion

Throughout the project, several models were compared with respect to the RMSE score (the lower the RMSE the better the model) to create an algorithm for the movie recommendation system.

First, linear models (Model 2, 3, 4) were tested, to which additional information (movie, user, release year) was added based on the analysis of the data set. The next two models (Model 5, 6) used a regularization parameter that penalizes movies and users with a small number of ratings. Final results for linear models is 0.8645223 (8 Final Validation for Model 6 Regularization m+u+r) which is improvement with respect to the baseline (Model 1 Baseline) which is just the average of all observed ratings.

Better results than linear models were achieved by the matrix factorization model, which gave the project the best result and the lowest RMSE value of 0.7828681 (9 Final Validation for Model 7 Matrix Factorization).

To improve the prediction results further, an ensemble method that combines different models can be considered to advance the recommendation system as much as possible.

5 Appendix

Project related documentation (MovieLens.R, MovieLens_report.Rmd, MovieLens_report.pdf) can be accessed on the GitHub https://github.com/matej-s/MovieLens . The MovieLens data set used in the project were downloaded from the https://grouplens.org/datasets/movielens/10m/ ,

The project used a Windows 10 computer with an i3 processor and 8 GB of RAM. The program code is written in R (version 4.0.2) and RStudio (version 1.3.1073) was used for development. Information collected on the session performed:

sessionInfo() R version 4.0.2 (2020-06-22) Platform: x86_64-w64-mingw32/x64 (64-bit) Running under: Windows 10 x64 (build 18363)

Matrix products: default

Random number generation: RNG: Mersenne-Twister Normal: Inversion Sample: Rounding

locale: [1] LC_COLLATE=English_United Kingdom.1252 LC_CTYPE=English_United Kingdom.1252
[3] LC_MONETARY=English_United Kingdom.1252 LC_NUMERIC=C
[5] LC_TIME=English_United Kingdom.1252

attached base packages: [1] stats graphics grDevices utils datasets methods base

loaded via a namespace (and not attached): [1] Rcpp_1.0.5 rstudioapi_0.11 knitr_1.29 magrittr_1.5
[5] tidyselect_1.1.0 munsell_0.5.0 colorspace_1.4-1 R6_2.4.1
[9] rlang_0.4.7 fansi_0.4.1 dplyr_1.0.2 tools_4.0.2
[13] grid_4.0.2 gtable_0.3.0 xfun_0.16 cli_2.0.2
[17] htmltools_0.5.0 ellipsis_0.3.1 yaml_2.2.1 digest_0.6.25
[21] assertthat_0.2.1 tibble_3.0.3 recosystem_0.4.3 lifecycle_0.2.0 [25] crayon_1.3.4 purrr_0.3.4 ggplot2_3.3.2 vctrs_0.3.2
[29] evaluate_0.14 glue_1.4.1 rmarkdown_2.3 compiler_4.0.2
[33] pillar_1.4.6 generics_0.0.2 scales_1.1.1 pkgconfig_2.0.3

For the described system, generating a pdf document (MovieLens_report.pdf) using the MovieLens_report.Rmd script takes about 3.5 hours, with the most time-consuming, about 90%, falling on the parts related to matrix factorization (2.4.7 Model 7 - Matrix Factorization and 3.2 Final validation for Model 7 Matrix Factorization).