MovieLens Project

Ilse Tromp

2024-07-16

MovieLens Project

Introduction

In this project, the MovieLens dataset will be used to develop an algorithm that can serve as a movie recommendation system with an RMSE that will be as low as possible. MovieLens is a site run by the University of Minnesota. The data set that will be used for this project is the 10M dataset, consisting of 10 million ratings and 100,000 tag applications on 10,000 movies by 72,000 users.

Loading the data set

Before we get started we first load some packages.

```
library(tidyverse)
library(caret)
library(ggplot2)
library(dplyr)
library(lubridate)
library(treemapify)
library(reshape2)
library(recosystem)
library(knitr)
```

Now we download the files that contain the data.

```
dl <- "ml-10M100K.zip"
if(!file.exists(dl))
  download.file("https://files.grouplens.org/datasets/movielens/ml-10m.zip", dl)

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

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

Next, we make sure our ratings variable is split into several variables, that these columns have the right names, and each variable is correctly defined as integer or numeric.

```
ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
```

```
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))
```

Now, we do the same for the movies variable. Only movieID has to be defined as integer.

```
movies <- as.data.frame(str_split(read_lines(movies_file), fixed("::"), simplify = TRUE),
                        stringsAsFactors = FALSE)
colnames(movies) <- c("movieId", "title", "genres")
movies <- movies %>%
  mutate(movieId = as.integer(movieId))
```

We can combine the data tables containing ratings and movies.

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

We will split our data set into a training and test set, so we can test in the end how well our algorithm works.

```
set.seed(1, sample.kind="Rounding")
```

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

```
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

Now we make sure this final test set also contains userID and movieID.

```
final_holdout_test <- temp %>%
  semi_join(edx, by = "movieId") %>%
  semi_join(edx, by = "userId")
```

Now, we add the rows back into our edx data set that we removed whilst creating our final test set.

```
removed <- anti_join(temp, final_holdout_test)
```

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

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

Finally, we transform the timestamp to a date:

```
edx <- edx %>%
mutate(date = as_datetime(timestamp))
```

The edx dataset consists of 9000055 rows and 6 columns. These columns represent the following parameters: userID (for each individual who has rated movies), movieID (an ID for each movie), rating (the rating a movie received from a user), timestamp (time in seconds since 01-01-1970), title (the title of the movie plus the year the movie came out), and genres (to which genres a movie belongs).

During this project, an algorithm will be developed that will estimate the rating (the variable that we would like to predict) based on the movie, genres, and user.

Exploratory data analysis

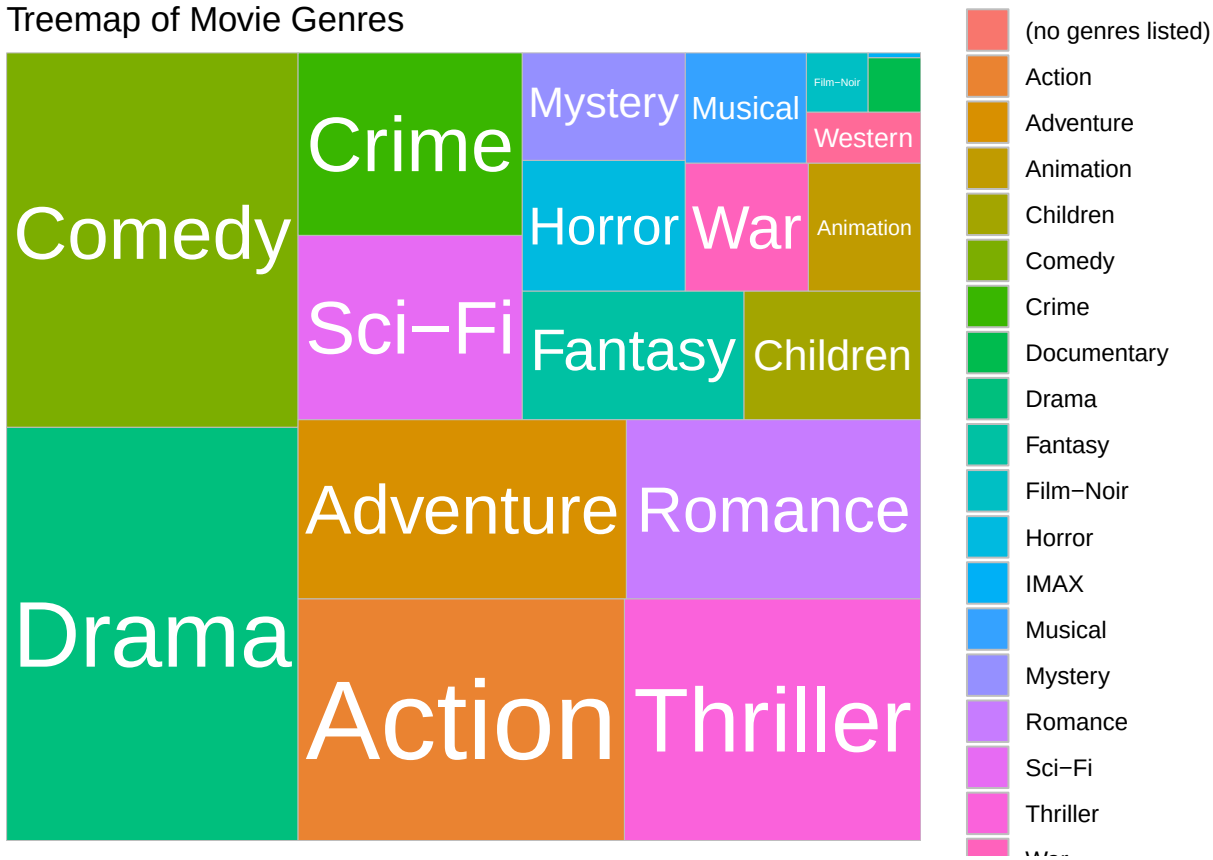
We will start with analyzing the data the gain some insights.

The number of movies per genre is:

```
##    Drama   Comedy Thriller  Romance
```

```
##  3910127  3540930  2325899  1712100
```

This treemap is a visualization of the movies per genre:



The most rated movie is:

```
## # A tibble: 1 x 2
##   title               count
##   <chr>               <int>
## 1 Pulp Fiction (1994) 31362
```

When accounting for ratings, the top ten best rated movies with at least 10,000 ratings are:

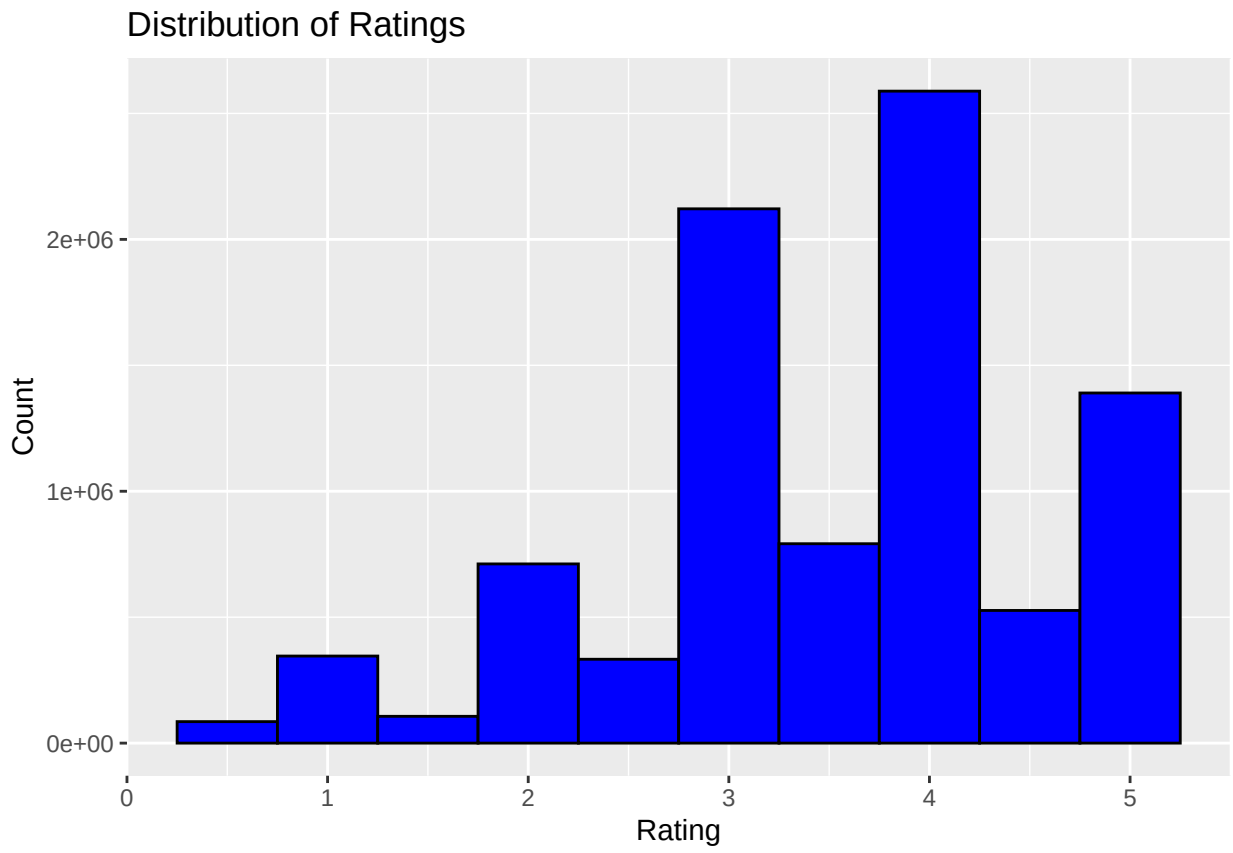
```
## # A tibble: 10 x 3
##    title                                                 average_rating     n
##    <chr>                                                          <dbl> <int>
##  1 Shawshank Redemption, The (1994)                                4.46 28015
##  2 Godfather, The (1972)                                           4.42 17747
##  3 Usual Suspects, The (1995)                                      4.37 21648
##  4 Schindler's List (1993)                                         4.36 23193
##  5 Casablanca (1942)                                               4.32 11232
##  6 Godfather: Part II, The (1974)                                  4.30 11920
##  7 Dr. Strangelove or: How I Learned to Stop Worrying and ~        4.30 10627
##  8 One Flew Over the Cuckoo's Nest (1975)                          4.29 13014
##  9 Raiders of the Lost Ark (Indiana Jones and the Raiders ~        4.26 19678
## 10 Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (~        4.22 25672
```

When accounting for ratings, the top ten worst rated movies with at least 10,000 ratings are:

```
## # A tibble: 10 x 3
```

```
##    title                                          average_rating     n
##    <chr>                                                   <dbl> <int>
##  1 Ace Ventura: When Nature Calls (1995)                    2.58 10430
##  2 Waterworld (1995)                                        2.86 14446
##  3 Batman Forever (1995)                                    2.92 17414
##  4 Dumb & Dumber (1994)                                     2.94 16053
##  5 Addams Family Values (1993)                              2.98 10061
##  6 Ace Ventura: Pet Detective (1994)                        2.98 18959
##  7 Home Alone (1990)                                        3.06 13800
##  8 Cliffhanger (1993)                                       3.08 13721
##  9 Star Wars: Episode I - The Phantom Menace (1999)         3.10 14125
## 10 Broken Arrow (1996)                                      3.12 12103
```

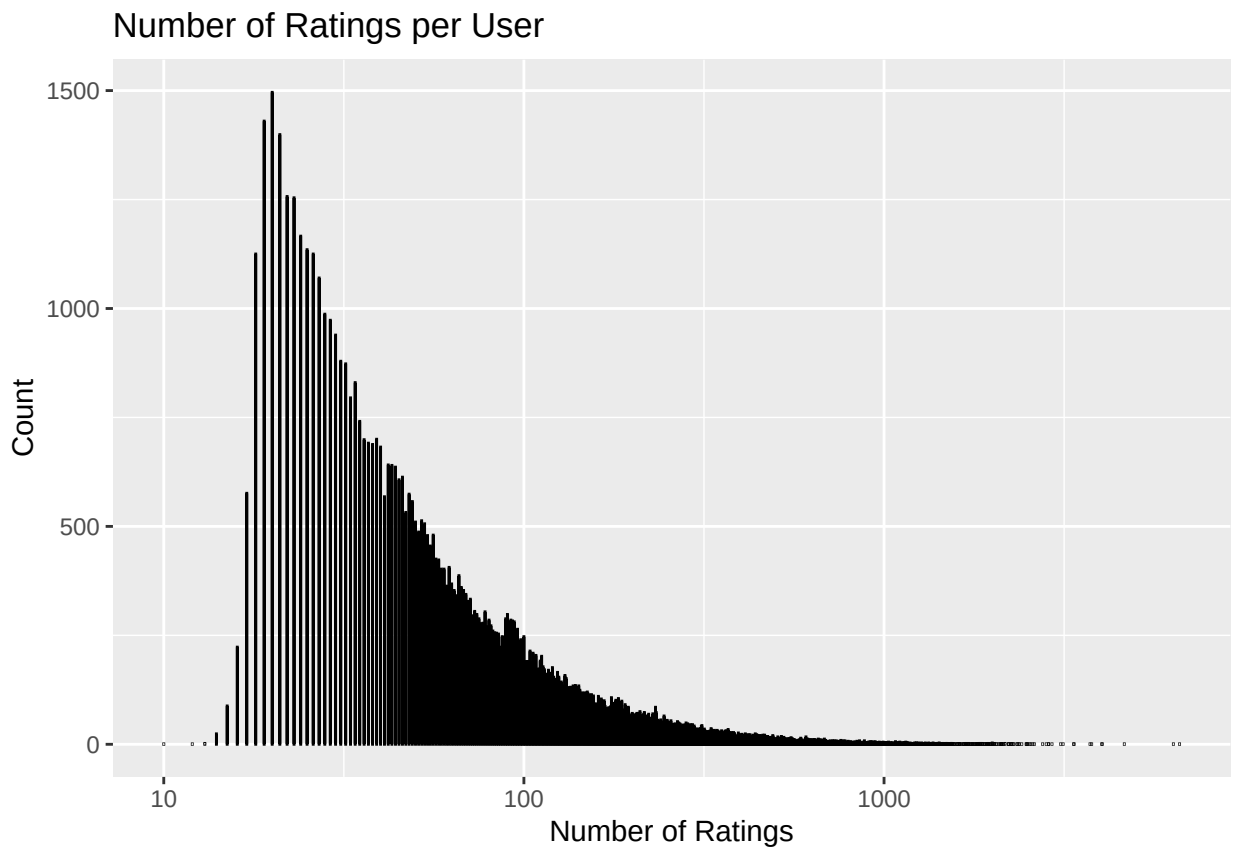
The distribution of ratings can be viewed in this plot:



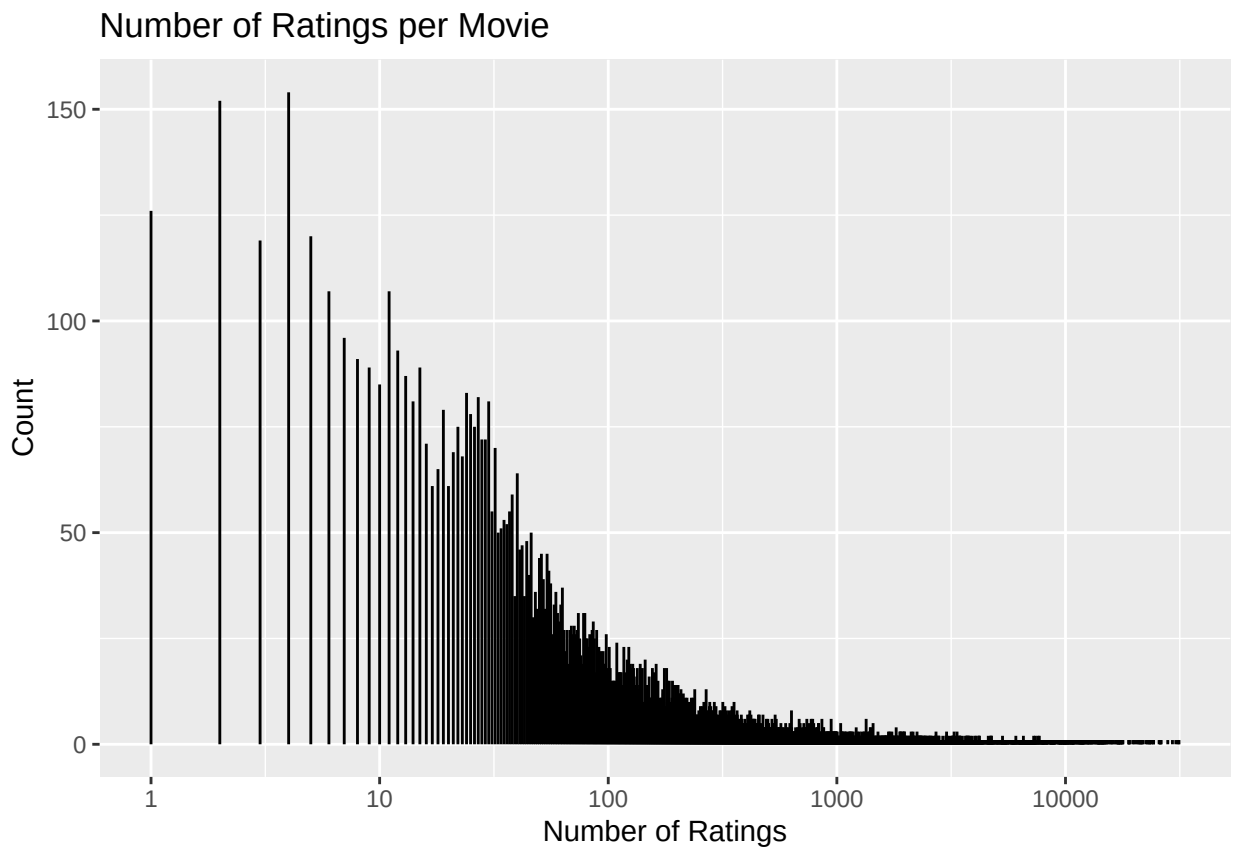
You can see that whole ratings (1, 2, 3, 4, and 5) are given more often than half ratings (0.5, 1.5, 2.5, 3.5, and 4.5).

Average user ratings:

The number of ratings per user can be viewed in this plot:



The number of ratings per movie can be viewed in this plot:



Data handling

When checking the data for missing values,

```
sum(is.na(edx))
```

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

```
colSums(is.na(edx))
```

```
##    userId   movieId    rating timestamp     title    genres      date
##         0         0         0         0         0         0         0
```

No missing values are found. So, there is no need to account for missing values.

Model development

In search for the best possible algorithm, several models will be assessed and evaluated. First, a baseline model will be build to compare the more advanced models to.

Baseline model

For this baseline model, the average rating across all movies

The average rating is:

```
## [1] 3.512465
```

The baseline model has the equation:

$Y = \mu + \varepsilon$

In code this looks like:

```
baseline_model <- rep(mu, nrow(final_holdout_test))
```

RMSE of this baseline model is:

```
## [1] 1.061202
```

This is a quite poor RMSE, so with the following models we will try to improve this.

Regularization

When making a model using both the userId and movieId effect, we can also account for total variability of movie and user effects by adding penalties. The formula for the penalty would be:

$$\sum_{i,j}(y_{u,i} - \mu - \alpha_i - \beta_j)^2 + \lambda \sum_j \beta_j^2$$

With the formula for the values of Beta being:

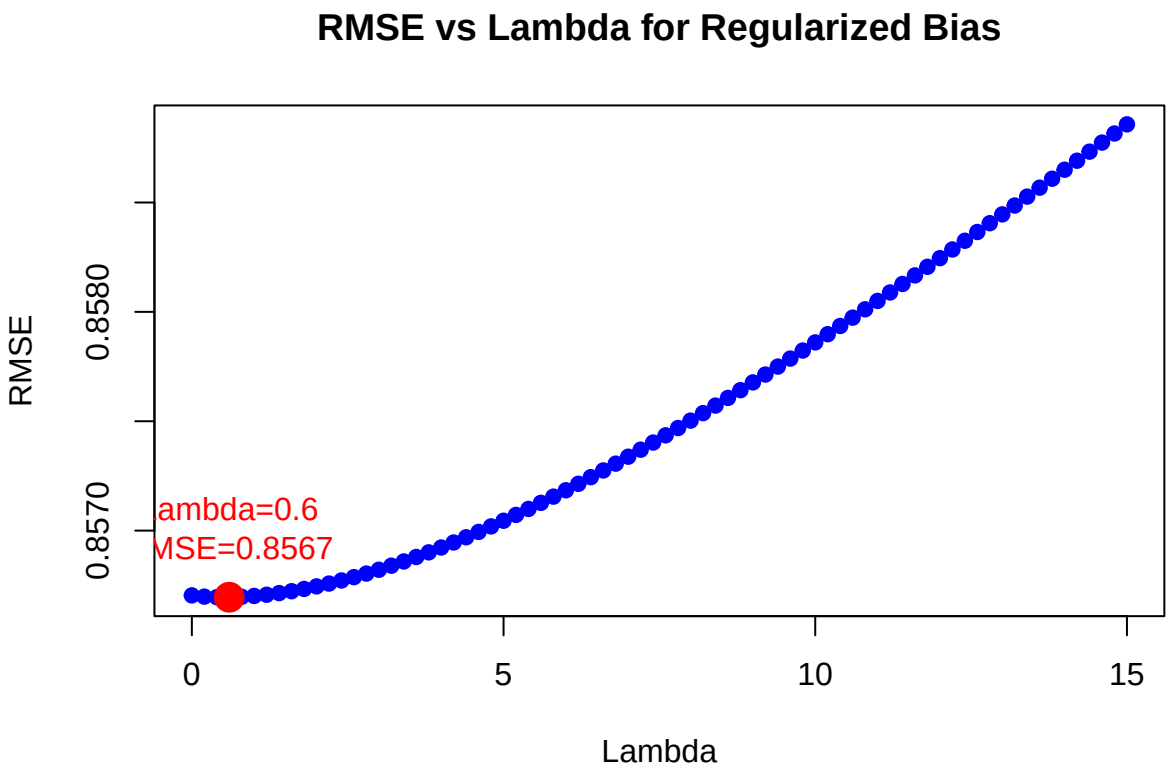
$$\hat{\beta}_j(\lambda) = \frac{1}{\lambda + n_j} \sum_{i=1}^{n_i}(Y_{i,j} - \mu - \alpha_i)$$

Before we can use this model, we need to calculate the optimal value for Lambda. We will calculate the RMSE for each value of Lambda and see which Lambda produces the lowest RMSE.

In this piece of code

```
Lambdas <- seq(0, 15, 0.2)

rmse <- sapply(Lambdas, function(Lambda){

  mu <- mean(edx$rating)

  a_i <- edx %>%
    group_by(movieId) %>%
    summarize(a_i = sum(rating - mu)/(n()+Lambda))

  b_u <- edx %>%
    left_join(a_i, by="movieId") %>%
    group_by(userId) %>%
    summarize(b_u = sum(rating - a_i - mu)/(n()+Lambda))

  modelled_ratings <- edx %>%
    left_join(a_i, by = "movieId") %>%
    left_join(b_u, by = "userId") %>%
    mutate(pred_rating = mu + a_i + b_u) %>%
    pull(pred_rating)

  return(RMSE(modelled_ratings, edx$rating))

})
```

We can plot the Lambdas against each RMSE



In the plot, we can see that the lowest RMSE is with Lambda:

```
## [1] "Best Lambda: 0.6"
```

```
## [1] "Best RMSE: 0.856695440471883"
```

Matrix factorization

The next model we will use is matrix factorization. The specific type of matrix factorization we will use, is called alternating least squares (ALS). This method should be suited for working with large datasets, like the edx data set. The package we will use to perform ALS is the recosystem package. However, before we can use the reco function, we need to prepare the data and select the variables we need.

```
edx_ratings <- edx %>% select(userId, movieId, rating)
test_ratings <- final_holdout_test %>% select(userId, movieId, rating)
```

Since I am working on an older laptop, I am saving the data into temporary text files to manage memory size.

```
write.table(edx_ratings, file = "training_data.txt", sep = " ", row.names = FALSE, col.names = FALSE)
write.table(test_ratings, file = "test_data.txt", sep = " ", row.names = FALSE, col.names = FALSE)
```

Now, we initialize the reco model and load the training data.

```
reco <- Reco()

training_data <- data_file("training_data.txt")
```

Next, we use ALS to train the model by using the reco$train function. We start using moderate values for the parameters to prevent overfitting of the model. We will check how this model performs. If it underfits, we can adjust these values.

```
reco$train(training_data, opts = list(dim = 20, lrate = 0.1, costp_l2 = 0.01, costq_l2 = 0.01, niter = 1
```

```
## iter      tr_rmse          obj
##    0       0.9613   9.0649e+06
##    1       0.8537   7.3760e+06
##    2       0.8023   6.6462e+06
##    3       0.7760   6.2961e+06
##    4       0.7601   6.0915e+06
##    5       0.7497   5.9599e+06
##    6       0.7425   5.8687e+06
##    7       0.7372   5.8022e+06
##    8       0.7331   5.7504e+06
##    9       0.7298   5.7079e+06
##   10       0.7270   5.6734e+06
##   11       0.7245   5.6425e+06
##   12       0.7224   5.6169e+06
##   13       0.7205   5.5940e+06
##   14       0.7187   5.5719e+06
##   15       0.7171   5.5525e+06
##   16       0.7156   5.5342e+06
##   17       0.7141   5.5170e+06
##   18       0.7127   5.5016e+06
##   19       0.7114   5.4862e+06
```

We can load the test data and make predictions by using the reco$predict function.

```
test_data <- data_file("test_data.txt")

predictions <- tempfile()
reco$predict(test_data, out_file(predictions))
```

```
## prediction output generated at /var/folders/rz/6b789ym956v7wzrxj4zm628h0000gn/T//RtmpDIx3Hf/file519e
```

```
predicted_ratings <- scan(predictions)
```

Lastly, we combine the actual and the predicted ratings into a data frame and calculate the RMSE.

```
predicted_ratings_df <- cbind(test_ratings, predicted_rating = predicted_ratings)

RMSE_matrix_factorization <- RMSE(predicted_ratings_df$rating, predicted_ratings_df$predicted_rating)
```

Matrix factorization using ALS results in an RMSE of:

```
## [1] "Matrix Factorization RMSE: 0.794423136894828"
```

This RMSE shows that the model has probably not under or over fitted.

Results and evaluation

During this project, we investigated three different models: the baseline model, a regularization model, and a matrix factorization model.

The different RMSEs of the different models can be viewed in this table:

Table 1: RMSEs of All Investigated Models

Model	RMSE	Parameters
Baseline Model	1.0612018	mu

Model	RMSE	Parameters
Regularization Model	0.8566954	Lambda = 0.6
Matrix Factorization Model	0.7944231	N.A.

The baseline model had an RMSE of 1.06, which is quite poor. The next model we tested was the regularization model by accounting for both user en movie effect. This model performed better with an RMSE of 0.857. The last model we tested was the matrix factorization model with ALS. This model performed the best with an RMSE of 0.793.

Conclusion

The regularization and matrix factorization models performed significantly better compared to the baseline model. The matrix factorization model's RMSE was even better than the winning RMSE of the Netflix challenge in 2009.

During this project, limited models were tested due to computing power. If more computing power would have been accessible, methods such as random forest, K-nearest neighbours, linear models, or ensemble models could have been tested.

Nevertheless, the results obtained during this project show a significant improvement of the RMSE.

References

https://movielens.org/info/about

Irizarry, Rafael A., "Introduction to Data Science: Data Analysis and Prediction Algorithms in R"

Koren, Y., 2009. The BellKor Solution to the Netflix Grand Prize.