Final Proyect

Liam Montero Guillemi

2025-03-19

Contents

Introduction **2**

What will the project do? . . . 2

Why is a good recommendation system important? . . . 2

How does it benefit businesses and users? . . . 2

Creating the dataset **3**

Creating the model **5**

How did I find which model to use? . . . 5

Implementation of the model . . . 5

Conclusions **7**

Introduction

What will the project do?

In this project, we want to create a movie recommendation system, in which we will use RMSE as a measure of loss.

Why is a good recommendation system important?

A good recommendation system is crucial because it allows new users to discover personalized movies from the start, increasing their satisfaction with the platform. It facilitates the discovery of content beyond what is popular, exposing users to options that truly interest them. This not only improves the user experience but also plays a fundamental role in long-term retention by offering continuous value through relevant suggestions. Ultimately, an effective system is key to the platform's success and competitiveness.

How does it benefit businesses and users?

A good recommendation system benefits both businesses and users. For businesses, it boosts retention, increases revenue, provides valuable user data, and offers a competitive advantage. For users, it facilitates personalized movie discovery, reduces information overload, exposes them to new content, and improves the overall platform experience, saving them time and increasing satisfaction. In essence, it creates a mutually beneficial relationship by connecting users with the content they love and businesses with more engaged audiences.

Creating the dataset

This part of the project was provided by the Edx platform, below we will explain step by step each line of it. The following lines of code install the tidyverse and caret packages if they aren't already installed. They are then loaded using the library() function.

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

Then we have the following 3 lines of code, which perform the following actions: 1) download the 10 million data document from the MovieLens dataset 2) extract the ratings data from the downloaded file 3) extract the movie name 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)
```

After that, the data is converted into a format that is easy to work with, in this case it is a data frame, everything that is inside the ratings_file file is read, then each of its rows is divided into columns using the separator "::"

```
ratings <- as.data.frame(str_split(read_lines(ratings_file), fixed("::"), simplify = TRUE),
                         stringsAsFactors = FALSE)
```

then each of these columns is named as follows:

```
colnames(ratings) <- c("userId", "movieId", "rating", "timestamp")
```

The data frame is configured so that each of the columns has the data in the desired format:

```
ratings <- ratings %>%
  mutate(userId = as.integer(userId),
         movieId = as.integer(movieId),
         rating = as.numeric(rating),
         timestamp = as.integer(timestamp))
```

We do the same thing as we did with the ratings_file, but this time with the movies_file, that is, we convert the data to an easy-to-work format, name its columns, and configure the data to a desired format:

```
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))
```

The 2 data frames created previously are joined using the movieId column:

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

Now, we proceed to divide the data set into a training set and a test set:

```
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
test_index <- createDataPartition(y = movielens$rating, times = 1, p = 0.1, list = FALSE)
edx <- movielens[-test_index,]
temp <- movielens[test_index,]
```

We make sure that all data in the test set is also in the training set as follows:

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

The rows that were removed from the test set are added to the training set so that this information is not lost:

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

Finally, all objects created to shape the training and test sets but no longer used are deleted from the workspace. This is done to free up memory space and allow for more comfortable work.

```
rm(dl, ratings, movies, test_index, temp, movielens, removed)
```

Creating the model

How did I find which model to use?

I'd like to start by saying that I decided to use the matrix factorization method because it offers the best results on large datasets. I decided to use the recosystem package and not the recommenderlab package because the recosystem package is optimized for large datasets, while the recommenderlab package is not. I learned this through trial and error. With the recommenderlab package, my computer never managed to finish the model; it always shut down before finishing it. I spent many hours running calculations. I think an Intel Core i9-13980HX isn't a bad processor, and paired with 32GB of RAM, it's not bad at all. Therefore, I decided to change my approach and use recosystem, which is designed to work with large datasets. It can perform parallel operations using the CPU, which greatly shortens processing times. It can load the training and prediction sets from the hard drive as a file, which means it doesn't overload the RAM too much. Although it also has the option to load them from the R workspace.

Implementation of the model

The first thing I did was install and load recosystem

```
if(!require(recosystem)) install.packages("recosystem")
if(!require(parallel)) install.packages("parallel")
library(recosystem)
library(parallel)
```

Then, we adjust the training set with only the parameters we are going to use, this is not absolutely necessary, but I like to do it to be more organized.

```
train_set <- edx %>% select(userId, movieId, rating) %>% data.frame()
```

We convert the training set into a format that the recosystem ecosystem can interpret, for this we use the data_memory() function, which uses an object that is in the R workspace and converts it into a DataSource object, which is the type of object used to train and predict models with the recosystem library.

```
train_data <- data_memory(train_set[, 1], train_set[, 2], rating = train_set[, 3])
```

creation of a RecoSys object, which will contain all the parameters, training and predictions.

```
r <- Reco()
```

Next, a hyperparameter adjustment is made, this is carried out by the r$tune function, which has a large list of many parameters to adjust, the vast majority of the parameters I chose were those that came by default, this because with chatGPT I did a search for the best parameters for large data sets and this gave me an answer that was very similar to the parameters that came by default. In the parameter nthread = detectCores() - 1, I put it like this because what this parameter does is use all the cores of your CPU except 1, and the parameter nbin = 32, is because it always has to be greater than the nthread parameter, so, since my CPU has 32 cores, nthread = 31 and nbin = 32.

This function performs cross-validation, which has 5 folds by default (which can also be adjusted), and runs all possible combinations of all parameters to find the best combination of them that results in a lower RMSE, if I wanted to evaluate using another loss function, I could also do so by adjusting the loss parameter, which has the RMSE by default.

```
best_tune <- r$tune(train_data, opts = list(dim = c(10L, 20L, 30L),
                                costp_l1 = c(0, 0.1),
                                costp_l2 = c(0.01, 0.1),
                                costq_l1 = c(0, 0.1),
                                costq_l2 = c(0.01, 0.1),
                                lrate    = c(0.01, 0.1),
                                nthread  = detectCores() - 1,
                                nbin     = detectCores())
)
```

Here I proceed to train the recommendation model with r$train, using the best parameters found with the r$tune function

```
final_model <- r$train(train_data, opts = best_tune$min)
```

Then, we proceed to prepare the evaluation data set, just as we did with the test data set, but this time we do not include the ratings, because that is what I am supposed to predict.

```
test_data <- data_memory(final_holdout_test[, 1], final_holdout_test[, 2])
```

We make predictions with r$predict and the object obtained is a vector, which contains all the ratings predicted by the model for the users.

```
pred2 = r$predict(test_data, out_memory())
```

Finally, we proceed to calculate the RMSE, to see the accuracy of our model, we do this using the vector of real values of the evaluation set and the vector obtained from the model's prediction on the evaluation set. General form of the least squares loss formula:

$RMSE < -sqrt(mean((true.values - predict.values)^2))$

```
RMSE <- sqrt(mean((final_holdout_test[, 3] - pred2)^2))
RMSE
```

```
## [1] 0.7823087
```

Conclusions

This project has enabled the development of a movie recommendation system based on matrix factorization techniques, demonstrating the efficiency of machine learning methods applied to large datasets. The most important points are highlighted below:

• **Data Preprocessing and Organization:** A rigorous extraction and transformation process was implemented for the Movielens dataset, ensuring proper data structuring from files with millions of records. The division into training and test sets, along with data integrity validation, was essential to build a solid foundation on which to train and evaluate the model.

• **Choice of Tools and Methodology:** The decision to use the recosystem package instead of recommenderlab was based on the need to handle a large volume of data and on computational efficiency. Recosystem enabled hyperparameter optimization through cross-validation and leveraged parallelization, resulting in a significant reduction in processing times and more efficient management of hardware resources.

• **Reflections and Future Directions:** This project demonstrates the importance of integrating data science techniques with robust optimization methodologies for handling large volumes of data. It also lays the groundwork for future improvements, such as the incorporation of new algorithms, further hyperparameter optimization, or adaptation of the system to other domains. The experience gained demonstrates that the combination of good preprocessing, the choice of appropriate tools, and an iterative approach to validation can lead to highly effective solutions in real-world environments.