Introduction

For years, many companies have tried to build a better recommendation algorithm from the MovieLens data set. As part of the Havard Data Science program, we have also decided to introduce ourselves to the recommendation algorithm world by working on MovieLens to propose a model that will outperform the existing ones.

There are many different MovieLens data sets, each different in size: 32 MB, 1MB, 1B, 100K, 1M, 10M, 20M, 25M. In this project, we will work with the MovieLens 10M data set. It was released in 1/2009 and consists of 10 million ratings and 100.000 tag applications applied to 100.000 movies by 72.000 users. The MovieLens 10M data set provided for this project was split into two.

- **EDX**, which contains 75% of the dataset. It will be used to train a model and determine the best one.
- **The Final Holdout Test**, which contains 25% of the data set. It will be used for prediction purposes only.

In the real-world project, the data scientist is provided only with the EDX data set, and the Final Holdout Test data set is nonexistent and unknown. The Final Holdout Test is the data set used to compare each competitor's performance in a competition. It is only kept by the organizer and accessible to the competitors. When they submit their model, it is evaluated with the Final Holdout Test, and the score is displayed on the leader board. The Final Holdout Test can also be seen as user interactions with the movie in real-time, and the model built should predict the rating that will be given to that movie. This project is almost similar to the 1st situation.

To obtain the best model, we will follow these steps

- Feature engineering: Create new features
- Because of the large size of the EDX, instead of using cross-validation, we will split it into two
  - training data set to train the model
  - test data set: Use to compare our different model
- Enumerate a list of algorithms or existing libraries that can be used. For each of them
  - Train the model with the training data set
  - Use the test data set to find the best hyper-parameter
  - Keep the hyper-parameter with the best RMSE
- Compare all the different models and keep the best one.
- Run the best model on the Final Holdout Test and save the RMSE.

Method/Analysis

Recommendation system algorithms

In machine learning, this problem is in the class of Recommendation Systems.

To solve a recommendation system problem, there are two approaches:

- Content-base filtering
- Collaborative filtering

In content-based filtering, we predict the rating a user will make on a movie based on the features of the user (age, location, etc. . .) and/or the feature of the movie (title, release date, genre, etc. . .). Either we use the features of the user to find similar users to him and predict his rating of the movie as a function of the rating of users similar to him, or we use the features of the movies to find similar movies to the movie the user wants to predict and predict his rating on that movie as a function of his rating on those similar movies.

In collaborative filtering, we predict the rating a user will give a movie based on what similar users have given or how he is rating movies similars to the one he wants to rate. Here, the focus is on the rating, not on the features.

Both of them work based on the computation of similarity. The main difference is that, in content-based filtering, the similarity between users or movies is sought based on their characteristics, while in collaborative filtering, the similarity is sought based on the rating (two users are similar according to how they rate a set of movies. two movies are similar to how a user rated them)

Our process/approach

Remember that the data was split in two:

- EDX for training, finding a machine learning model
- Final Holdout test to determine how efficient the model **Question**: How to build that model

In the content-based filtering approach, we need:

- to know the features of the users and the movies. Unfortunately in this project, we don't have any user features, but we do have features for movies.
- to build the model: there is no model but a process. A model is built from training and reused on a test set. In content-based filtering, we have a process to implement when looking for the rating of a user on a movie

In the collaborative filtering approach, there are two main approaches:

- Memory-based: It applies statistics to all data sets to make predictions. To find the rating R of user U to movie M,
  - Find users similar to U who have rated movie M
  - Calculating the rating R based on the ratings of users found in the previous step
- Model-based: It consists of reducing or compressing the large but sparse user-movie matrix through dimensionality reduction. The goal is to find the best A and B matrix such that they can compute together to have the initial user-movie matrix.

In this project, we have decided to use the **collaborative filtering (model-based)** approach. We decided to follow that approach because it responds to the implicit process given to us, process defined by the two data sets provided to us: The training set and the Holdout test set. They suggest that we should find a model from the training set (EDX) and check how the model is good from the holdout test (The Final Holdout test set).

The model-based approach of collaborative filtering involves finding two matrix A and B from the user-movie matrix also called the rating matrix R such that the product of the two matrix A and B gives us the approximative user-movie matrix R.

$$Finding A\ n * k\ and B\ m * k\ such that R = AxB^T$$

The real A and B do not exist. An algorithm called matrix factorization approximates the user-movie matrix R with A and B.

The training will consist of finding the best A and B so that their product is the closest to R based on the RMSE.

Multiple parameters intervene in the search for the best A and B. The main one is the dimensions of A and B. Some algorithms find A and B, such as A (nU x k) and B (k x nM). So, the training will consist of finding the best K that reduces the RMSE. K is called the number of factors. R = A x B. Other algorithms find A and B such that A (nU x nU) and B (nM x nM) with a matrix in the middle of size (nU x nM) R = A x E x B

We will implement the Matrix factorization approach, which consists of finding the best A and B such that we have R = A x B + u + bU + bM According to whether or not we include the bias bU and bM, we have two models.

Also, there is the Hybrid matrix factorization, which consists of matrix factorization based on a user's rating of a movie but also taking into consideration the features of users and movies. It uses content-based and collaborative data.

Taking into consideration the users/movies bias and the users/movies features, we have

Bias	Features	Models
True	True	Model-ByFy
True	False	Model-ByFn
False	True	Model-BnFy
False	False	Model-BnFn

In R programming, there is a library called `cmfrec`, which can handle those 4 cases. In each of them, we will only focus on finding the excellent hyperparameter k such that A and B will be the best in terms of smallest `RMSE`.

Data preparation

We starts by preparing the data, mainly feature engineering.

- Extracting movie release date from movie name column
- Multiple hot encoding from genre column And after that, we split the data to into convenient data sets to use in finding the best models among the 4 defined up there.

Multiple-hot encoding on Genre

The `edx` dataset contains a column named `genres`. A movie can belong to multiple genres, and there are 18 different movie genres. Handling string is complicated and required huge preprocessing. One simple preprocessing data we can do, is the multiple-hot encoding.

```
# list of all movie genres
movie_genres <- c("Action", "Adventure", "Animation", "Children", "Comedy",
                  "Crime", "Documentary", "Drama", "Fantasy", "Film-Noir",
                  "Horror", "Musical", "Mystery", "Romance", "Sci-Fi", "Thriller",
                  "War", "Western")

genres <- edx$genres

split_genre <- function(genre) {
  s <- unlist(str_split(genre, fixed("|")))

  return(as.integer(movie_genres %in% s))
}
# Create a data.frame with 18 columns and same number of rows than edx
genres.splitted <- do.call(rbind, lapply(genres, FUN=split_genre))

# Naming the columns with the Genre name
colnames(genres.splitted) <- movie_genres
```

Extract year release date

The `edx` dataset contains a column named `title,` which includes the movie's name and release year. We extract the release year and use it as a movie feature like this.

```r
log_info("Extract [year] & [title] from movie [name]")

movies.data <- genres.splitted
movies.name.splitted <- edx %>%
  mutate(
    year = as.integer(str_extract(title, "(?<=\\()\\d+(?=\\)$)")),
    name = str_squish(str_remove(title, " \\(\\d+\\)$"))
  ) %>% select(name, year)
movies.data <- cbind(movies.data, year = movies.name.splitted$year)
```

After creating those new features, we can create a new data frame `edx.data` containing only features that we will use in training the model

```r
log_info("Create a new edx data frame with the previously created movie data")
edx.data <- cbind(edx, movies.data) %>%
  select(-c(title, genres))
```

Split data

The data for training is reading. As we have to find the best K, the process consists of splitting the training data set `edx.data` into training and validation data sets. The training data set will be used to create the model, and the validation dataset will be used to evaluate the obtained model. The process is always used for finding the best hyperparameters.

We will split our `edx.data` data set into 80% for training and 20% for validation.

```r
set.seed(1, sample.kind="Rounding") # if using R 3.6 or later
# set.seed(1) # if using R 3.5 or earlier
validation_index <- createDataPartition(y = edx.data$rating, times = 1, p = 0.25, list = FALSE)
train.edx <- edx.data[-validation_index,]
temp <- edx.data[validation_index,]

# Make sure userId and movieId in final hold-out validation set are also in edx set
validation.edx <- temp %>%
  semi_join(train.edx, by = "movieId") %>%
  semi_join(train.edx, by = "userId")

# Add rows removed from final hold-out validation set back into edx set
removed <- anti_join(temp, validation.edx)
train.edx <- rbind(train.edx, removed)

rm(validation_index, temp, removed)
```

Results

For each of our models, we ran the `CMF` algorithm, with good parameters for each value of K from 3 to 30. For each K, we train the model using `X.train` and `X.train.movies` if needed to obtain a model and then compute the validation error from `X.validation`. In the end, we plot the RMSE as a function of K and determine the best value of K.

Conclusion