Movie recommendation system

Student
Marco Plazzogna

1 Introduction

1.1 Context

This project consists in developing two Python Classes, which we are going to call **MovieRecommender** and **GenreRecommender**. The objective is to create a system that, given two datasets (the first with movies' info and the second with the ratings), produces a recommendation for movies or genres based on the the ratings which other users gave on a scale from 1 to 10. In order to accomplish this task, we are going to use a correlation index between the ratings of several movies given by various reviewers. Later, we are going to do something slightly different with the genres as well.

The project focuses on the student's ability to build classes and methods at a higher level of abstraction and to do some basic data cleansing for the datasets used.

1.2 Datasets

The datasets are taken from `https://grouplens.org/datasets/movielens/latest/`: approximately we are dealing with 33,000,000 ratings applied to 86,000 movies by 330,975 users (last updated 9/2018).

The first dataset **movies.csv** contains the following variables:

- moviesId: integer number;
- title: string of the movie title (usually it comes with the year of release);
- genres: string describing the genre of the movie.

The second dataset **rating.csv** contains the following variables:

- userId: integer number;
- movieId: integer number;
- rating: integer number from 1 to 10 (originally from 0.5 to 5 with a step of 0.5).

2 Methodology

2.1 Preliminary analysis and data cleansing

Since **movies.csv** serves only as an information dataset, we focused more on the other one. Firstly, we removed the column "timestamp", since it is not interesting for our analysis and then checked for invalid ratings. Then, we thought that having a look at the distribution of the unique ratings might be of some interest to be sure that we have reasonable data and not strange values. Other kind of operations could have been done, like checking the mean or the standard deviation for the overall ratings or filtered by type of genre, but the main point is to be able of building the classes in a later stage of the project.

Then, we proceed to a harsher part of the project. Clearly, the number of ratings and movies is huge, resulting in some problems with the rating matrix that we want to build in order to compute later the correlation.

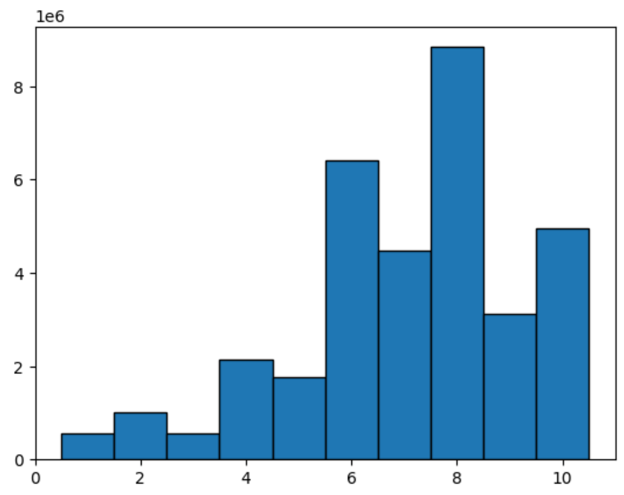

Figure 1: Distribution of the ratings

Listing 1: Dimension in Gb of the original rating matrix

```
    n = len(np.unique(ratings['movieId']))
    m = len(np.unique(ratings['userId']))
    print([n, m, (32 * n * m) * 1.073e-9])
```

```
[83239, 330975, 945.9577622663999]
```

To manage this problem, we used the int8 format from the Numpy library and reduced the number of ratings and movies thanks to the method **_create_rating_matrix(self)** from the Class **MovieRecommender** (for more details on this method, look at the code). In this way, we managed to have a rating matrix, no larger than 8 Gb (dimension chosen arbitrarily and it can be modified to be bigger or smaller). Similar operations were done for the Class **GenreRecommender** that we are going to describe better in **Methods implemented in the class GenreRecommender**.

2.2 Methods implemented in the class MovieRecommender

- **__init__(self, rating, movie)**: initialises the class, the arguments required are the two datasets. It initialises several attributes, some of them calling other methods such as **_create_rating_matrix(self)** and **_title_to_id_to_title(self)**;
- **_create_rating_matrix(self)**: creates the zero matrix with shape adjusted for a reasonable amount of memory occupied (set 8Gb but modifiable for other needs);
- **set_rating(self)**: sets all the ratings of the rating matrix;
- **get_rating_matrix(self)**: returns the rating matrix (practically it functions as a print);
- **get_shape(self)**: returns the shape of the rating matrix;
- **plot_rating_matrix(self)**: returns the plot of the rating matrix;
- **_title_to_id_to_title(self)**: returns a dictionary whose keys are the ids (movieId) for the corresponding values which are the titles of the movies (title);
- **get_name_movies(self)**: returns a list with name of all the movies considered for the recommendation system (more useful for testing);
- **__call__(self, movie_name)**: implements the recommendation system with correlation function defined previously. It requires a movie_name to suggest the first 5 recommended movies (improvable, look at the **Comments on MovieRecommender** section for further informations). It also returns a plot for the the first 1000 correlation indices.

2.3 Methods implemented in the class GenreRecommender

- **__init__(self, rating, movie)**: initialises the class, the arguments required are the two datasets. It initialises several attributes, some of them calling other methods such as **_create_genre_matrix(self)**, **_adapt_genre(self)** and **_genre_to_id_to_genre(self)**;
- **_adapt_genre(self)**: method needed to adapt the **rating** dataset with a new column corresponing to the genre;
- **_create_genre_matrix(self)**: creates the zero matrix with shape adjusted for a reasonable amount of memory occupied (set 8Gb but modifiable for other needs). The main difference from the **_create_rating_matrix(self)** method from the **MovieRecommender** class is the use of float16 (instead of int8) because in this case we have to handle some numbers that are means;
- **set_rating(self)**: sets all the ratings of the rating matrix;
- **get_genre_matrix(self)**: returns the rating matrix (practically it functions as a print);
- **get_shape_genre(self)**: returns the shape of the rating matrix;
- **plot_genre_matrix(self)**: returns the plot of the rating matrix;
- **_genre_to_id_to_genre(self)**: returns a dictionary whose keys are the ids (movieId) for the corresponding values which are the genres of the movies (genres);
- **get_name_genres(self)**: returns a list with name of all the genres considered for the recommendation system (more useful for testing);
- **__call_genre__(self, genre_name)**: implements the recommendation system with correlation function defined previously. It requires a genre_name to suggest the first 5 recommended genres. It also returns a plot for the the first 1000 correlation indices.

3 Results

3.1 MovieRecommender Class

Here we decided to report only the most important results. For further informations or notes, we suggest to look directly at the code.

For example, here we print the rating matrix, where the blue areas represent the nonzero part of the matrix:

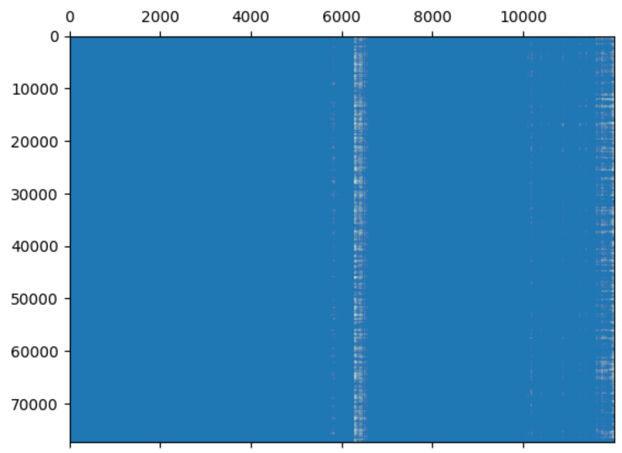

Figure 2: Rating matrix for users and movies

Here there are the first five suggestions for the movie 'Sudden Death (1995)' and the plot for the correlation index for the first 1000 recommended movies:

```
    print(recommender.__call__('Sudden Death (1995)'))
```

```
["Big Girls Don't Cry (Große Mädchen weinen nicht) (2002)", 'SherryBaby (2006)',
 'In-Laws, The (1979)', 'Wog Boy, The (2000)', 'Pursuit of Happyness, The (2006)']
```

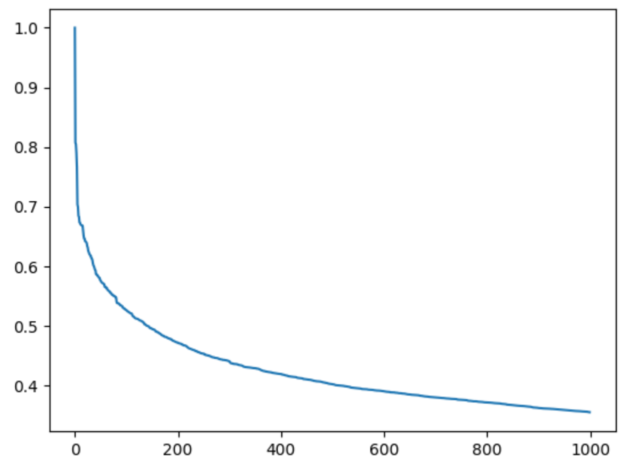

Figure 3: Graph of the first 1000 correlations for the movie 'Sudden Death (1995)'

3.2 Comments on MovieRecommender

We want to report a limitation that can eventually be improved in the method **__call__(self, movie_name)**: it does not accept values that are only similar to the original name from the dataset. For example:

```
print(recommender.__call__('Sudden␣Death'))
```

```
The movie Sudden Death was not found
```

Even though, the year might be important because of the homonymy of two movies in different periods, it may be more useful to find a movie given an incomplete title. This can clearly be upgraded, but was left as a small detail to report to focus on other aspects of the project considered more relevant.

The last detail to underline is the level of abstraction in the class. In fact, we assume that the datasets given, already have some columns named the way we want them to be (e.g: 'title', 'genres' for **movies.csv** or 'rating', 'movieId' for **ratings.csv**). This is not a problem for our project, but generally, inside a class there should be no reference to known objects outside of it.

3.3 GenreRecommender Class

Now, we would like to see some results in the **GenreRecommendation** class. Firstly, we report the rating matrix related to the genres (next page):

Then, we intend to explain some lines of code that we used to build the new data frame:

```
self.new_rating = pd.concat([self.rating,
    pd.DataFrame(self._adapt_genres())], axis=1)
self.new_rating.columns = ['userId', 'movieId', 'rating', 'genres']
self.genre_mean = self.new_rating.groupby(['userId',
    'genres'],as_index=False)['rating'].mean()
self.genre_mean = pd.DataFrame(self.genre_mean)
```

In the first two lines we combine the original dataset **ratings.csv** and the corresponding genres taken from the **movies.csv** dataset thanks to the **_adapt_genre(self)** method. Then, in the third and in the fourth lines, we proceed to compute the means filtered by 'userId' and 'genres', and finally build the new dataset we are going to work with.

These are the first 5 suggestion for the genre 'Comedy|Romance' and the first 1000 correlation indices:

```
print(recommender2.__call_genre__('Comedy|Romance'))
```

```
['Comedy', 'Drama|Musical', 'Drama', 'Action|Adventure', 'Comedy|Drama']
```

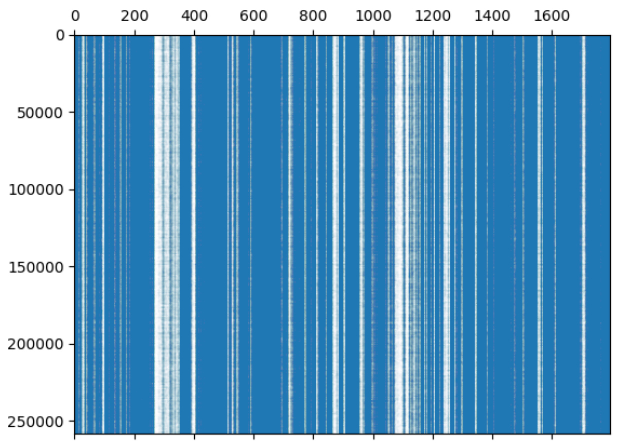

Figure 4: Rating matrix for users and genres

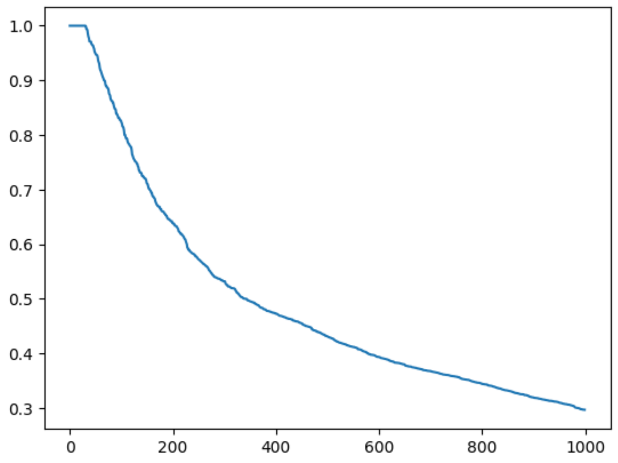

Figure 5: Graph of the first 1000 correlations for the genre ' Comedy | Romance '

3.4 Comments on GenreRecommender

The correlation function used in the second class is the same one used in the first one, but with different rating matrices and shape.

Since the two classes are very similar in some senses, we would like to comment on the same things as before such as the abstraction of the class and on the problems related to the name of the genre used in the **__call_genre__(self, genre_name)** method.

4 Conclusion

In the end, we can state that we created two functioning classes for movie and genre recommendation, satisfying the objectives of the project. There are of course some limitations, already described in **Comments on MovieRecommender** and in **Comments on GenreRecommender** which can be definitely improved. However, it's a basic and functioning recommendation system, based on very large data and with the implementation of some functions from Pandas, Numpy and Matplotlib. So we can conclude that the guidelines of the project are fulfilled.