# Lab Session 10: Surprise for Python recommenders

**Alex Carrillo Alza, Roger Creus Castanyer**
Cerca i Anàlisi de la Informació, Ciència i Enginyeria de Dades (UPC)

December 9, 2019

## 1 Introduction

### 1.1 Nearest-neighbor-based recommender

Firstly, this implementation is straightforward as this algorithm is directly derived from a basic nearest-neighbors approach. More specifically, we use a basic collaborative filtering algorithm, taking into account the mean ratings of each user, `KNNWithMeans()`:

$$\hat{r}_{ui} = \mu_u + \frac{\sum\limits_{v \in N_i^k(u)} \text{sim}(u,v) \cdot (r_{vi} - \mu_v)}{\sum\limits_{v \in N_i^k(u)} \text{sim}(u,v)} \qquad \text{or} \qquad \hat{r}_{ui} = \mu_i + \frac{\sum\limits_{j \in N_u^k(i)} \text{sim}(i,j) \cdot (r_{uj} - \mu_j)}{\sum\limits_{j \in N_u^k(i)} \text{sim}(i,j)}$$

depending on the chosen approach: user-based or item-based. In this case, similarities will be computed between users by default.

It should be noted that the recommender is set biased, which means it corrects the prediction using user averages and item averages, makes predictions more stable if the user or item has few ratings, and it also allows to give some decent prediction if user or item have no ratings.

### 1.2 SVD-based recommender

In the case of the matrix factorization implementation, it maps users and items to a joint factor space based on the *singular value decomposition* theorem (SVD):

$$\mathbf{M}_{(n \times m)} = \mathbf{U}_{(n \times k)} \cdot \mathbf{\Sigma}_{(k \times k)} \cdot \mathbf{V}^{\top}_{(k \times m)} \qquad \text{with} \qquad \mathbf{n} \text{ users, } \mathbf{m} \text{ items and } \mathbf{k} \text{ factors}$$

The first nuance is that we moderate the number of factors to $k = 10$ in order to better interpret results. Furthermore, the recommender is set biased as in the previous one, for the same reasons. After fitting the model, the `qi` attribute of the predictor is generated, which each column of it tells how much each item is related to a factor, *i.e.* the $\mathbf{V}_{(m \times k)}$ matrix.

## 2 Experimenting

### 2.1 Knn with new users

Once we have implemented both recommenders with code, it is possible to predict the rating of a given user for a given item (movie), as long as the user belongs to the ratings dataset. That adoption of it is not useful, as in practice we would like to predict the rating that a user who is not in the database would give to a movie that he/she has not seen.

After several trial and test errors we noticed that the only way to achieve that is by adding the users in the `ratings.csv` file by hand, with the whole information like that:

| | |
|---|---|
| 611,5,5.0,1493846415 | 612,14,5.0,1493846415 |
| 611,18,5.0,1493846415 | 612,26,5.0,1493846415 |
| 611,19,5.0,1493846415 | 612,40,5.0,1493846415 |
| 611,65,5.0,1493846415 | 612,55,5.0,1493846415 |
| 611,104,4.0,1493846415 | 612,57,5.0,1493846415 |
| | 612,147,5.0,1493846415 |
| | 612,148,4.0,1493846415 |

were `611` is a new user with a set of 5 *comedy* movies and `612` is a new user with a set of 7 *drama* movies, with really high ratings. Now, by reading the data as we did before, we can train both models with the added information of this users and ask for the predicted ratings for a movie that they have not seen.

For instance, using the $k$-nearest-neighbor recommender and trying some movies of different topics we get the following results:

| **611** | | **612** | |
|---|---|---|---|
| *movieid*: 924 | *pred. rating*: 5.0 | *movieid*: 924 | *pred. rating*: 5.0 |
| *movieid*: 54 | *pred. rating*: 4.17 | *movieid*: 54 | *pred. rating*: 4.31 |
| *movieid*: 595 | *pred. rating*: 4.98 | *movieid*: 595 | *pred. rating*: 5.0 |

As it can be seen we get a predicted rating for any given movie that is in the database. However, we noticed that the majority of ratings are above 4.0, which indicates, in our opinion, that as there is a sheer volume of movies and the user movies list is not that large, when we ask for only one recommendation it returns the biggest one from a very long list of high related movies.

## 2.2 SVD factorization

Now we use the new users presented above but with the SVD recommender. This way, we will have a better understanding of how factors affect the predicted ratings. The recommender, which is also biased, is built with 10 factors, so we deal with the matrix of rank 10 that better approximates the *user* $\times$ *movie* matrix (this one contains the ratings).

We interpret that those 10 factors contain each a component of the whole variability of the ratings distribution. These could be the separation that users with different preferences show on rating films by its topic.

When experimenting we show the top 10 most relevant films for each factor, and even though there is no way that the factors can be tagged we observe that theme-related films tend to contribute stronger to a common latent factor, and even more if this film is only described by 1 topic.

```
$ [(1979 Star Wars:  Episode I - The Phantom Menace (1999), 0.49599801888249395),
$ (1674 A Space Odyssey (1992), 0.44469674589514324),
$ (6746 Taken (2008), 0.40545098390552703),
$ (2507 Mission to Mars (2000), 0.3764646053152664),
$ (302 Ace Ventura:  Pet Detective (1994), 0.3727507971501269)]
```

After many experiments, in some outputs we found films like "Star Wars", "Mission to Mars" and "A Space Odyssey" being grouped together with some other other films also tagged as "Sci-fi". Many other curious examples were found, although in some cases it feels like trying to fit a pattern in our minds because of the need to find an answer to ratings distribution.

In general, in terms of execution time, the matrix factorization take longer than the knn with means. And the index matching with the movie id's between different files doesn't help at all.