M. Benjamin Negrevergne

Collaborative Filtering based on Clustering data preprocessing

Authors:
Salomé Papereux
Ryan Belkhir
Tom Duval

Abstract

Here is the report for the first assignment on Recommender Systems using Matrix Factorization. We quickly realized that classical methods to solve this problem such as Stochastic Gradient Descent or Alternating Least Square allow to obtain very good results on data sets of different sizes. This is the reason why we decided to look at clustering methods (hard/soft) to perform a pre-processing consisting in gathering users who look alike and perform a matrix factorization on each cluster. We will therefore discuss the different observations and results we have obtained with these methods and future avenues to explore.

October 24, 2021

Contents

1 Introduction

The success of the movie industry since the early 40's is supported by the development of on-demand movie platforms in the 2000's. Faced with competition and the concern for customer satisfaction, movie recommendation systems have become an essential strategy in this industry, when the choice offered to users is ever more varied. For this first project, we worked on matrix factorization for collaborative filtering recommender system, applied to the Movie Lens dataset. On the first hand, we focused on the basic method, to understand the how it works, its advantages and its limitations. We computed different factorizations to compare the impact of the decomposition on the accuracy of the models. On the second hand, we thought about a method that could improve score predictions, based on users' identity. This second approach groups users and movies into clusters based on indicators such as age, user occupation, or movie genre. This idea is supported by the research of "The Role of Gender and Sensation Seeking in Film Choice" [2] : "this study indicate that movie themes [...] clearly interact with gender in complex ways and thereby affect movie viewing preference". Through this method, we want to find out to what extent the characteristics of users and/or movies influence preferences and, therefore, the prediction of ratings.

Let's consider R the ratings matrix, of n users and m movies, thus $R \in \mathbb{R}^{n \times m}$. R_ij corresponds to the rate of user i for movie j, rate between 1 and 5, or 0 if the movie isn't rated. The objective of MF is to reconstruct the ratings matrix R, by factoring it by two matrices, which we call user-matrix U and movie-matrix M. Thus R $\sim$ UM, where $U \in \mathbb{R}^{n \times k}$, $R \in \mathbb{M}^{m \times k}$. The mathematical formulation of this problem corresponds to solve:

$$\min_{U,M} \|R - UM\|_{\mathcal{F}}^2 + \lambda(\|U\|_{\mathcal{F}}^2 + \|M\|_{\mathcal{F}}^2) \tag{1}$$

To compute the performance of the models we decided to use the RMSE. This metric is a good indicator of prediction accuracy. By applying the same score calculation on all methods, it allows us to compare the efficiency of the methods between them. Moreover, since the R rating matrix is sparse, we computed the RMSE only on the strictly positive values, i.e., the movies rated by the users. This allows us not to bias our accuracy results, since the 0s, replacing the missing entries, do not correspond to scores. The calculation of the RMSE metrics is:

$$RMSE = \sqrt{\frac{\sum\limits_{(i,j)\in S} R_{ij} - (UM^T)_{ij}}{|S|}} \; with \; S = \{(i,j) : R_{ij} > 0\} \tag{2}$$

We use several methods to solve this optimization problem such as: Gradient Descent, Stochastic Gradient Descent, Alternate Least Squares and Singular Values Decomposition (SVD).

2 Optimizations of classical Matrix Factorization methods

2.1 Gradient Descent and Stochastic gradient descent

We first computed Gradient Descent (GD) and Stochastic Gradient Descent (SGD). Their objective is to minimize (P_min) by annulling its derivative with respect to U and M. At each iteration, we update U and M, until reaching convergence of the RMSE. Here's a more precise mathematical calculation of GD approach:

$$\text{For all } t \geq 0 : \begin{cases} M_{t+1} = M_t - \eta_t(-2RU + 2MU^TU + 2\lambda M) \\ U_{t+1} = U_t - \zeta_t(-2R^TM + 2UM^TM + 2\mu U) \end{cases}$$

Then the associated RMSE is calculated. Here M_t and U_t are optimized simultaneously.

The method of SGD is very similar but differ while iterating. In GD, we calculate the loss and the derivative on all points simultaneously, whereas in SGD, we use single point in loss function, chosen randomly among other without replacement.

The difficulty encountered during the computation of GD was the complexity of execution. Since we compute

the RMSE without counting the missing entries, we performed the gradient descent note by note, to compute $R_{ij} - (UM^T)_{ij}$ for each pair $(i,j) \in S$. The idea was to reduce the triple loop (epochs, loop on i, loop on j), in order to reduce the computational time. We thus took advantage of the features of the numpy library to generalize the computations on integer rows, integer columns, and even up to the entire matrix, as for example to determine the indices of strictly positive scores in R. We went from an execution time of 4 seconds per epoch, to about 0.23ms per epoch.

About SGD, as the goal of the method is to take every pair $(i,j) \in S$ randomly for each epoch, we had a hard time to reduce the execution time. However, we improved it in the best way we could, without affecting the particularity of the SGD approach. An epoch has a duration of about 2.5 seconds.

2.2 Alternating Least Squared

We used an alternative approach, Alternated Least Squared (ALS). It is a two-step iterative optimization, which at every iteration, first fixes U and solves for M, and then fixes U and solves for P. It allows to optimize M independently from U and vice versa. This time, iterative steps are the following:

$$\text{For all } t \geq 0 : \begin{cases} M_{t+1} = (U_t^T U_t + \lambda I)^{-1} (U_t)^T R \\ U_{t+1} = (M_t^T M_t + \mu I)^{-1} M_t^T R \end{cases}$$

As M and U are optimized simultaneously, this operation reduces the problem to a linear regression: we find in the calculation of M_{t+1} and U_{t+1} the solution of the ordinary least squared formula.

In the same way as before, we have improved the complexity caused by the loops on the indices thanks to the numpy features.

2.3 Singular Values Decomposition

Another approach of matrix decomposition we used is truncated Singular Values Decomposition (SVD).
The objective is to factorize R by using truncated SVD of rank $k << \min(m,n)$. The matrix R is computed as $R = Q_k \Sigma_k P_k^T$, with $Q_k \in \mathbb{R}^{n \times k}, \Sigma_k \in \mathbb{R}^{k \times k}$ and $P_k^T \in \mathbb{R}^{m \times k}$. We defined the users' matrix $U = Q_k \Sigma_k \in \mathbb{R}^{m \times k}$ and movies' matrix $M = P_k^T \in \mathbb{R}^{m \times k}$.

Because the matrix R of ratings is sparse, the simple decomposition of R in three matrices $R = Q_k \Sigma_k P_k^T$ is biased. In order to deal with zeros values, we applied to R an algorithm which replaces missing values for a fixed user u by the mean of the rates of u. We used algorithm from source [1]. It does the following steps:

- Initialization step: Initialize the missing entries as following:

  $$R_{f,ij} = \begin{cases} R_{ij} \text{ if } R_{ij} \neq 0 \\ \text{mean}(R_i) \text{ without missing entries} \quad \text{if } R_{ij} = 0 \end{cases}$$

- Iterative steps:
  - Step 1: Perform rank-k SVD of Rf in the form $Q_k \Sigma_k P_k^T$
  - Step 2: Readjust only the originally missing entries of R_f to the corresponding values in $Q_k \Sigma_k P_k^T$.

  $$\text{Mathematically,} R_{f,ij} = \begin{cases} R_{ij} \text{ if } R_{ij} \neq 0 \\ (Q_k \Sigma_k P_k^T)_{ij} \text{ if } R_{ij} = 0 \end{cases}$$. Go to iterative step 1 until convergence.

We also applied Gradient Descent and ALS after initializing matrices U and M according to the rank-k SVD. The results aren't conclusive, because we have the same RMSE as the simple Gradient Descent and ALS methods. We conclude this new initialization has no effects on these methods. As mentioned above, we initially tried to factorize R

with the truncated SVD in one iteration, by mean centering the R matrix. However, the operation was not conclusive because the method is not adapted to sparse matrices, the missing values strongly biasing the predictions.

3 Motivations for using Unsupervised-Learning approaches

The matrix factorization is a process that allows to recommend movies to users, through the idea that there are significant similarities between users. This idea is transcribed through the mathematical calculations of the previous sections. For the rest of the project, we naturally thought of a more concrete and interpretable method, allowing to group users and movies according to their characteristics. If users are similar, then it is likely that their preferences are similar. Conversely, if movies are similar, it is likely that most of them are liked by the users. It is on these assumptions that we have built our previous model below.

We modeled this approach by grouping users into clusters, and movies into clusters. To be able to access the identity of the users, and the characteristics of the movies (in addition to the ratings), we used the older dataset of 1 million ratings released in 2003. There are 6000 users registered in the database for 4000 films. For the users, we chose to focus our clustering on gender, age, and occupation. As for the movies, the indicators we chose to form our clusters are the year of release, and the genre(s). We performed two forms of clustering which are the following:

- Hard-Clustering with the K-Means algorithm (User based, Movie based and User-Movie based)
- Soft-Clustering with the Gaussian Mixture Model

4 Collaborative Filtering using Clustering data preprocessing

4.1 K-Means

To do this, we decided to apply a clustering algorithm in data pre-processing.
We first thought of the K-Means algorithm but the data being categorical, we turned to the K-Mode algorithm.

Using the available data in the dataset *Movielens-1M*, we represented each user i by a vector u_i. The first element represents its gender (0: women, 1: man), the second one its age (ex: 18 for people between 18-24) and the last one its occupation (2 represents artists for example). In the same way, we represented the information for each movie with a vector of size 18 (each index is a category of movie), which has a value 1 if the movie belongs to the category, 0 otherwise.

To determine the best value of the hyper-parameter K we use the Silhouette and Elbow methods. We then performed three different clustering strategies which are the following:

- **User based**:

  Instead of solving a matrix factorization problem on a huge rating matrix R. We split R into k_{user} (number of user's cluster) smaller sub-matrices and solve the same problem for each of them.

  Let consider $R = \begin{pmatrix} 2 & 0 & 0 & 5 \\ 4 & 3 & 0 & 2 \\ 1 & 0 & 3 & 4 \\ 5 & 3 & 0 & 0 \end{pmatrix}$ and two clusters $C_1 = \{u_1, u_3\}, C_2 = \{u_2, u_4\}$,

  we then have to solve the two following matrix factorization on

$R_1 = \begin{pmatrix} 2 & 0 & 0 & 5 \\ 1 & 0 & 3 & 4 \end{pmatrix}$ and on $R_2 = \begin{pmatrix} 4 & 3 & 0 & 2 \\ 5 & 3 & 0 & 0 \end{pmatrix}$

- **Movie based**:

  In the same way, we divide the matrix R in k_{movies} smaller sub-matrices by considering the clusters on the columns and not on the rows.

- **User-Movie based**:

  Finally we tried to combine the two methods above in order to work on $k_{movies} \times k_{users}$ sub-matrices much smaller and whose elements between them (user or movie) are supposed to be similar.

We first implemented our own K-Means algorithm but after several failures in cluster creation. We discovered that it was not suitable for categorical data. Due to lack of time, we used the K-Mode algorithm of the eponymous library to perform the clustering. Once the clustering was done we had to manage the splitting of the matrix R in accordance with the clusters. Then we just had to use the matrix factorization algorithms previously implemented.

4.2 Gaussian Mixture Model

After having explored a hard clustering method (K-means) we will use a soft clustering method which is the *Gaussian Mixture Model* to conclude our research on recommendation systems. The name soft clustering is due to the fact that a user can belong to several clusters. A Gaussian Mixture Model is a density model where we combine a finite number of K Gaussian distributions $\mathcal{N}(x \mid \mu_k, \Sigma_k)$ so that

$$p(x|\theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x \mid \mu_k, \Sigma_k) \tag{3}$$

$$0 \leq \pi_k \leq 1 \ , \ \sum_{k=1}^{K} \pi_k = 1 \tag{4}$$

where we defined $\theta := \mu_k, \Sigma_k, \pi_k \ : \ k = 1, ..., K\}$ as a collection of all parameters of the model. Before expressing the Data Generation model we used to solve Matrix Completion with GMM, here are some elements of notation that we will use:

- $R \in \mathbb{R}^n$
- C_u: set of movie that user u has rated
- $x_i^{(u)} = \{x_i^{(u)} : i \in C_u\}$: vector of observed value

So our model is the following:

- GMM is generating $x^{(1)}, ..., x^{(n)} \in \mathrm{R}^n$
- K mixture components: $\theta := \mu_k, \Sigma_k, \pi_k \ : \ k = 1, ..., K\}$
- $p(x^{(u)}|\theta) = p(x_{C_u}^{(u)}|\theta) = \sum_{j=1}^{K} \pi_j \ p(X_{C_u}^{(j)} \mid \mu_{C_u}^{(j)}, I_{|C_u| \times |C_u|})$

We will use the EM-algorithm to obtain the best possible estimate of θ. The EM algorithm consists in finding the maximum likelihood estimator θ^{MLE} by maximising the following log-likelihood:

$$\hat{l}(X,\theta) = \sum_{u=1}^{n}\sum_{j=1}^{K} p(j|u)\ \log(\frac{\pi_j\ p(X_{C_u}^{(j)} \mid \mu_{C_u}^{(j)}, I_{|C_u|\times|C_u|})}{p(j|u)})\ \ with\ \ p(j|u) = \frac{p(x^{(u)}|j,\theta)\ \pi_j}{\sum_{j=1}^{K} p(x^{(u)}|j,\theta)\pi_j} \tag{5}$$

Finally, to complete the row $x_i^{(u)}$ for all values $i \notin C_u$ where C_u is the set of observed values, we have :

$$x_i^{(u)} = \sum_{j=1}^{K} p(j|u)\ \mu_i^{(j)} \tag{6}$$

To determine the best number of mixtures of our model we used a statistical criterion: the *Bayesian Information Criterion*. This criterion consists in capturing the best trade-off between the likelihood of the data and the number of parameters that the model uses. In a situation where we wish to select models, we want a model with the the highest BIC.

5 Results and discussions

Now that we have presented all the models we used and the motivation and approach we followed, we will present and explain our results.

Here is the evolution of the RMSE of the different models on the 100K MovieLens dataset:

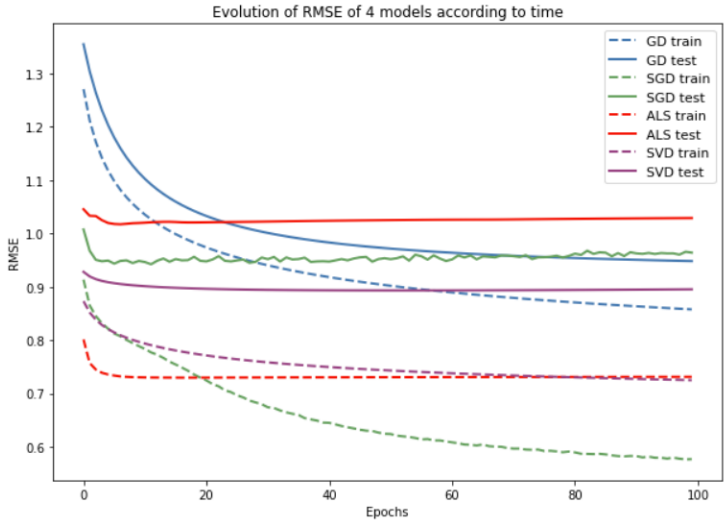

Figure 1: Evolution of RMSE of four different models according to time on the 100K MovieLens dataset

We observe that the SVD approach has the best RMSE among all model we have computed, even though it overfits. We believe that this difference of precision is caused by the fact that missing entries are initialized on the mean of each user, and the SVD is learnt on it. Another advantage of this method is the convergence speed according to the precision. The GD method is the 2nd best approach but less strong in convergence speed and accuracy. Considering the first epochs, the SGD method converges extremely quickly to an accuracy close to that of the SVD. However, its accuracy doesn't improve according to time, it's even the opposite. This method proves to be efficient for few iterations. Finally, even though ALS converges quickly, it is not as accurate as the others, and remains the least

accurate obtained. We could also have added a bias to reduce the overfitting of some methods but we preferred to focus on pre-processing the data via clustering methods.

See below for the optimal hyperparameters for all the methods we used on the 100K MovieLens dataset.

Epochs: 100	Learning rate	K (latent vector)	λ (Regularization term)	RMSE
Gradient Descent	10^{-4}	12	2	Train: 0.86
				Test: 0.93
Stochastic Gradient Descent	10^{-2}	10	0.05	Train: 0.57
				Test: 0.96
ALS		2	1.2	Train: 0.73
				Test: 1.02
SVD		3		Train: 0.72
				Test: 0.89

Table 1: Best hyperparameters for the several methods on the 100K MovieLens dataset

As we can see, applying clusters before performing the matrix factorization does not improve the results and even reduces them a bit. From the experiments that we have been able to carry out, this would come from some clusters that have an rmse of 4 or more on the test set where some others have a rmse of 0.27.
We believe that it is possible to improve models with clustering by further improving their hyperparameters tuning and refining the clusters. Finally, although it is not in the subject we think that this method can offer recommendation solutions for the "cold-start" problem.

We also observed that soft clustering methods (GMM) did not work as well as the classical methods mentioned above on huge sparse matrices. However we tried on less sparse matrices and the results were much more encouraging, we obtained RMSE of about 0.48. Moreover, since the initialization of the mixtures is random and the performance of the EM algorithm depends greatly on the initialization of the mixtures, there are many avenues to be explored to obtain better results via a GMM (using a K-Means for initialization for example).

References

[1] Charu C. Aggarwal. “Recommander Systems,” in: (2016).

[2] Marina Krcmar Zhanna Bagdasarov Dovile Ruginyte Smita C. Banerjee Kathryn Greene. “The Role of Gender and Sensation Seeking in Film Choice”. In: (2014).

[3] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: (2015).

[4] Data Science Made Simpler. “Alternating Least Squares”. In: (2015).

[5] Yan Tang Hangyu Yan. “Collaborative Filtering Based on Gaussian Mixture Model and Improved Jaccard Similarity”. In: (2019).

[6] Martin Haugh. “The EM Algorithm”. In: (2015).

[7] René Vidal Yi Ma and S.Shankar Sastry. “Generalized Principal Component Analysis”. In: (2016).

Annexe

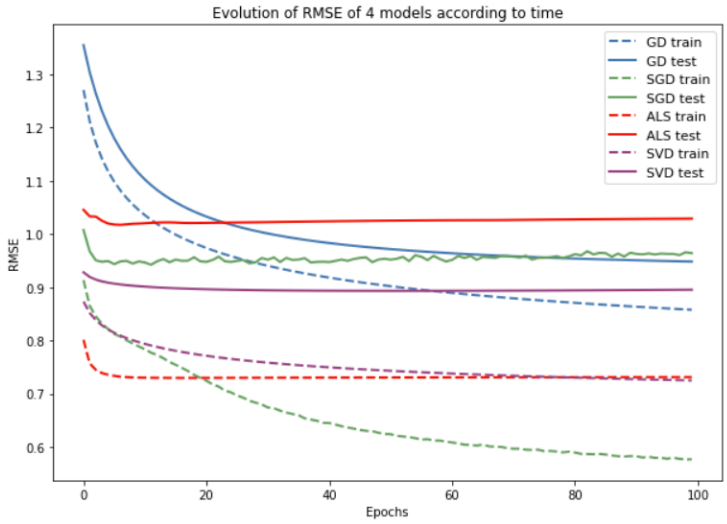

Figure 2: Optimal hyperparameters for all the methods we used on the 100K MovieLens dataset

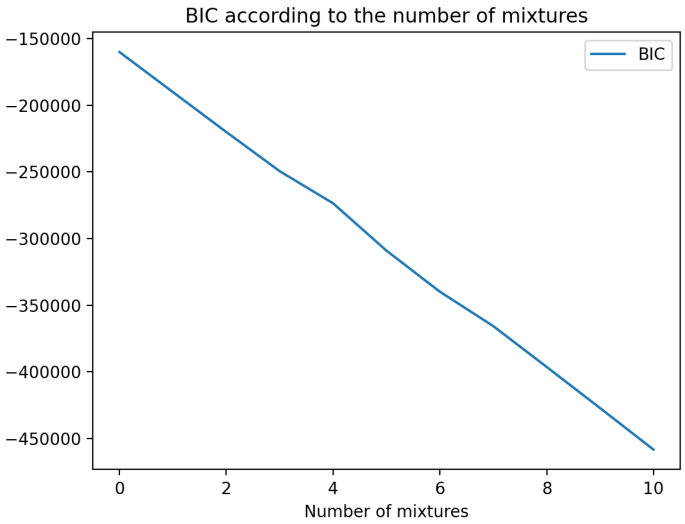

Figure 3: Evolution of the BIC according to the number of mixture

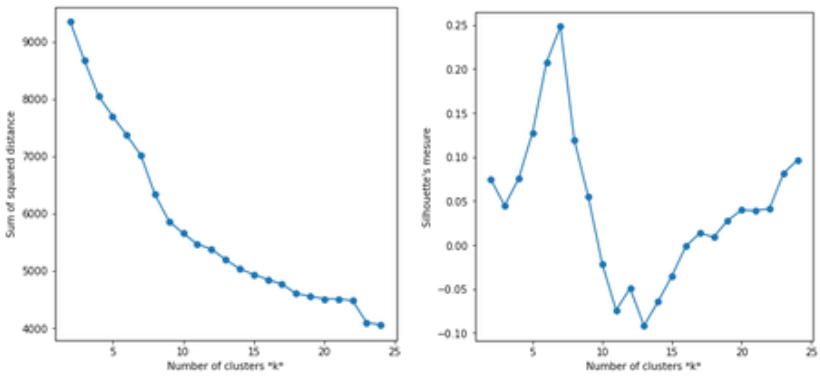

Figure 4: Elbow and Silhouette methods to find the best hyperparameter K in K-Mode (users)

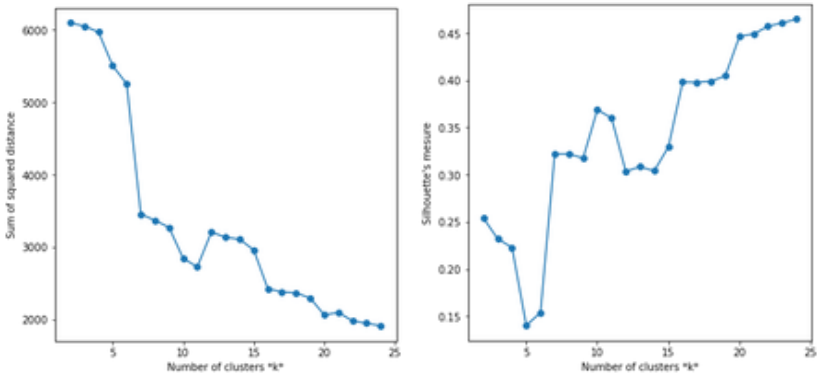

Figure 5: Elbow and Silhouette methods to find the best hyperparameter K in K-Mode (movies)

Epochs: 50	Learning rate	K (latent vector)	λ (Regularization term)	RMSE
Gradient Descent	10^{-4}	6	0.002	Train: 0.9007 Test: 0.9422
Stochastic Gradient Descent	10^{-2}	6	0.1	Train: 0.8478 Test: 0.8988
ALS		6	4	Train: 0.8099 Test: 0.8973
SVD		6		Train: 0.8256 Test: 0.9219
ALS with User-based clustering		6	4	Train: 0.7694 Test: 0.9331
ALS with Movie-based clustering		6	4	Train: 0.7261 Test: 1.0599
ALS with both clustering		6	4	Train: 0.7163 Test: 1.0457
SGD with User-based clustering	0.01	6	0.1	Train: 0.8834 Test: 0.9389
SGD with Movie-based clustering	0.01	6	0.1	Train: 0.8402 Test: 0.9610
SGD with both clustering	0.01	6	0.1	Train: 0.8177 Test: 0.9491

Table 2: Best hyperparameters for the several methods on the 100K MovieLens dataset