Probabilistic matrix factorization for recommender systems

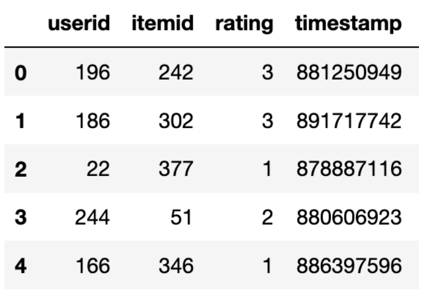
Caterina De Bacco

1 Matrix factorization: the problem

Imagine to observe a set of ratings that users make for a certain item, e.g. a movie they watch. How can you use this to predict whether the users will like a movie that they have not yet seen? In other words, how can we make good item recommendations? This is the general problem of recommender systems and we observe it in many daily situations, e.g. playlists of songs or movies to watch.

Objective: recommend user items that they may want to consume.

The data can be represented as a $N \times M$ matrix of entries $r_{ia} \in \mathbb{R}$ being the rate that a user i give to item a. Here N and M are the numbers of users and items, respectively, see Figure 1 as an example.



	userid	itemid	rating	timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596

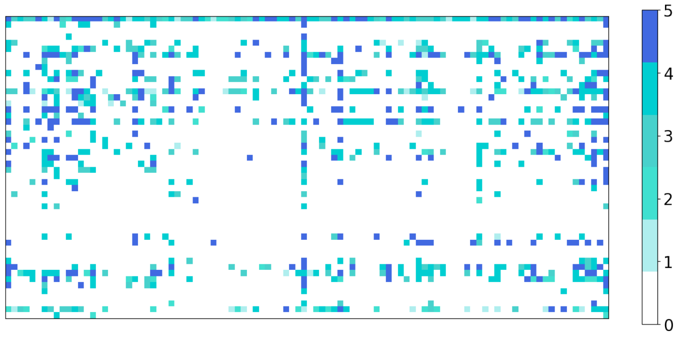


Figure 1: Example rating dataset in tabular format (left) and matrix (right).

A standard way to solve this issue is to look for a low-dimensional representation of this matrix, which can be obtained by solving the following optimization problem:

$$\min_{U,V} \sum_{i,a} (r_{ia} - \sum_k u_{ik} v_{ak})^2 + \lambda_u ||U||_2^2 + \lambda_v ||V||_2^2 \quad , \tag{1}$$

where U and V are the user and item matrices of entries u_{ik} and v_{ak}, respectively. They represent factors of dimension K that one can use to measure the users' preferences and the items' features. This is similar to what we saw in previous lectures about the stochastic block model, where we had membership vectors.

Obs1: so far the problem is deterministic, in the sense that we do not pose any probability distribution but need to proceed only using linear algebra. In particular, one can use singular value decomposition (SVD) to find optimal U and V. Alternatively, a local minimum of that objective function can be found by performing gradient descent or by solving an alternating least-squares problem in U and V.
For instance, one step of gradient descent is:

$$u_{ia} = u_{ia} + \gamma \left[(r_{ia} - u_i \cdot v_a) \cdot v_a - \lambda_u u_{ik} \right] \quad , \tag{2}$$

which only requires calculating derivatives of the cost function w.r.t. each individual parameter.

Here we take a probabilistic perspective, and model the likelihood of observing a rating matrix as:

$$P(R|U,V,\sigma) = \prod_{i,a} \left[\text{Gauss}(r_{ia}; u_i \cdot v_a, \sigma^2) \right]^{I_{ia}} \quad , \tag{3}$$

where $I_{ia} = 1$ if user i rated item a, 0 otherwise. Notice that this is similar to what we saw with Poisson factorization, but here we have real data, hence we use a Gaussian distribution instead.
We can then set Gaussian priors on the parameters as well:

$$P(U;\sigma_U) = \prod_i \text{Gauss}(u_i; 0, \sigma_U^2 \mathbb{I}) \tag{4}$$

$$P(V;\sigma_V) = \prod_a \text{Gauss}(v_i; 0, \sigma_V^2 \mathbb{I}) \quad . \tag{5}$$

With this, we have a log-posterior as:

$$\log P(U,V|R,\sigma,\sigma_U,\sigma_V) = \frac{1}{2\sigma^2} \sum_{i,a} I_{ia}(r_{ia} - u_i \cdot v_a)^2 - \frac{1}{2\sigma_U^2} \sum_i u_i \cdot u_i - \frac{1}{2\sigma_V^2} \sum_a v_a \cdot v_a \quad , \tag{6}$$

where we ignored contribution not depending on the parameters U and V.
By defining $\lambda_U = \sigma^2/\sigma_U^2$ and $\lambda_V = \sigma^2/\sigma_V^2$, this can all be re-written as:

$$E = \sum_{i,a} I_{ia}(r_{ia} - u_i \cdot v_a)^2 - \lambda_U \sum_i ||u_i||_2^2 - \lambda_V \sum_a ||v_a||_2^2 \quad . \tag{7}$$

This is the cost function observed in Eq. (1)! Hence, we see how using these choices of likelihood and prior gives a probabilistic interpretation to the non-probabilistic decomposition of Eq. (1).
Obs2: similar results could be obtained for the Poisson matrix factorization, but with a different norm (i.e. not the 2-norm as in here).
Exercise: what norm?

Once you have the U and the V you can use them to make recommendations. How?
Well, you can assign a score:

$$score_{ia} = \mathbb{E}[r_{ia}] = u_i \cdot v_a \tag{8}$$

to all unobserved ratings (i,a). You will then recommend a user i the items a with higher scores $score_{ia}$.

Obs3: the expected rating $score_{ia}$ may not be in the range of available scores. For instance, if a rating system as values $r_{ia} \in \{1,\dots,K\}$, the expected score is likely not to be in this range. A way to tackle this is to rescale the rating in a pre-processing step by using the function $t(x) = (x-1)/(K-1)$, so that the new rating is in $[0,1]$. One can then consider a modified model where the mean of the Gaussian likelihood is $g(u_i \cdot v_a)$, where $g(x) = 1/(1+\exp(x))$ is the logistic function. This may make calculations more difficult.
Obs3: with this probabilistic approach, users with low ratings will have predicted rating of unobserved items close to the average rating.

1.1 Adding biases and implicit feedback

Users may behave quite differently. For instance, certain users report higher ratings while other users may be more severe in their judgement, see Figure 2 for an example of this.

This may result in biased estimates, so how do we account for possible such biases? A possible way to deal with this heterogeneity is to add extra parameters that account for biases, so that the expected value of the rating is now:

$$\mathbb{E}[r_{ia}] = \mu + b_i + b_a + u_i \cdot v_a \quad , \tag{9}$$

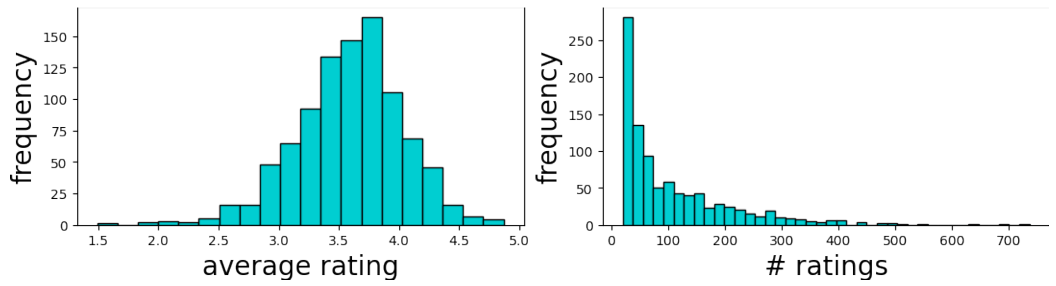


Figure 2: Average rating (left) and number of ratings (right) per user in MovieLens dataset.

where μ is a global bias, i.e. a certain dataset my have higher ratings than another one; b_i and b_a are user and item biases, respectively. The first one accounts for a user i to have higher ratings than other users, while the latter accounts for certain items receiving higher ratings in general.

Another possible issue is that certain users may not report that many ratings, compared to the number of items they consume, see for instance Figure 2 for an example of this.

Can we use somehow the information about what items they consumed but not rated to better estimate their ratings? Yes, by assuming that consuming an item is an implicit feedback on liking that item. One can integrate this assumption is various ways into the model. For instance, one can add an extra K-dimensional factor w_a to an item so that a user's factor u_i can be decomposed into:

$$u_i = u_i^0 + \frac{\sum_a I_{ia} w_a}{\sum_a I_{ia}} \quad . \tag{10}$$

In this way, we can get more accurate estimates of the ratings of users that report very few ratings.

2 Implicit data

What if we do not have access to any rating at all but only observe click or consumption data? We call this implicit data, similarly to above. This is usually easier to obtain, but harder to model... This is because it is usually binary data, hence we must use zero entries as well, i.e. unused items, to predict items that may be liked. If you replicate the same model as for the rating data (or explicit data), then you may treat an unrated item as a negative rating. But in these big datasets there a many unrated items simply because a user did not know about it. This becomes an important problem indeed in the presence of sparse datasets, which is often the case in recommendation systems. How can we deal with this problem?

The observed data is now a $N \times M$-dimensional matrix P with entries $p_{ia} = 1, 0$ if a user i consumed item a.

One popular approach Hu *et al.* (2008) is that of assigning confidence values c_{ia} to the rating r_{ia}. For instance:

$$c_{ia} = 1 + \alpha r_{ia} \quad , \tag{11}$$

where α is an hyper-parameter and has to be tuned with cross-validation. This assumes that the higher the positive rating, the higher the confidence of user i about item a. Now the new cost function becomes:

$$\min_{U,V} \sum_{i,a} c_{ia} (p_{ia} - u_i \cdot v_a)^2 + \lambda_U \sum_i ||u_i||_2^2 + \lambda_V \sum_i ||v_i||_2^2 \quad . \tag{12}$$

This has the effect of downgrading the zeros, and giving thus higher weight to the positive ratings.

Obs: notice that while the data is sparse, we need to use the whole $N \times M$ matrix P to estimate the parameters. Hence, the algorithm is not scalable to large system size. However, there could be ways to improve computational efficiency by using alternating least square suitable parallelized Hu *et al.* (2008).

3 Cross-validation and performance metrics

Assuming that you now have various algorithms to perform matrix factorization. How do you test which one is best? This can be done using cross-validation:

- split the dataset into train and test set;
- infer the parameters by fitting the algorithm on the training set;
- calculate performance metrics on the test set;
- choose the model that has highest average performance over all test sets.

One possible performance metric is the mean square error (MSE), as this is related to the cost function in Eq. (1):

$$MSE = \frac{1}{NM} \sum_{i,a} (r_{ia} - \hat{r}_{ia})^2 \quad , \tag{13}$$

where $\hat{r}_{ia}$ is the inferred rating matrix. In case you have a discrete rating matrix, a better metric can be the mean absolute error (MAE):

$$MAE = \frac{1}{NM} \sum_{i,a} |r_{ia} - \hat{r}_{ia}| \quad . \tag{14}$$

A part from recovering the true rating one can also test the models in their ability to recommend items that users may like. For this, appropriate metrics retrieval measures such as recall@k, precision@k, mean average precision (MAP@k) and Normalized Discounted Cumulative Gain (NDCG@k). All of these metrics use the scores in Eq. (8) to make a ranking of items that a user may like and counts how many of these are correct (in different ways, based on the metric).

Good references for this lecture are Koren *et al.* (2009); Mnih and Salakhutdinov (2008); Hu *et al.* (2008).

References

Y. Hu, Y. Koren, and C. Volinsky, in *2008 Eighth IEEE International Conference on Data Mining* (Ieee, 2008) pp. 263–272.

Y. Koren, R. Bell, and C. Volinsky, Computer **42**, 30 (2009).

A. Mnih and R. R. Salakhutdinov, in *Advances in neural information processing systems* (2008) pp. 1257–1264.