Published in Towards Data Science

Oscar Contreras Carrasco Follow
May 7, 2020 · 9 min read · Member-only · Listen

PMF for Recommender Systems

Probabilistic Matrix Factorization and Collaborative Filtering

Automated recommender systems are commonly used to provide users with product suggestions that would be of their interest based on existing data about preferences. The literature describes different types of recommender systems. However, we will highlight two major categories and then expand further on the second one:

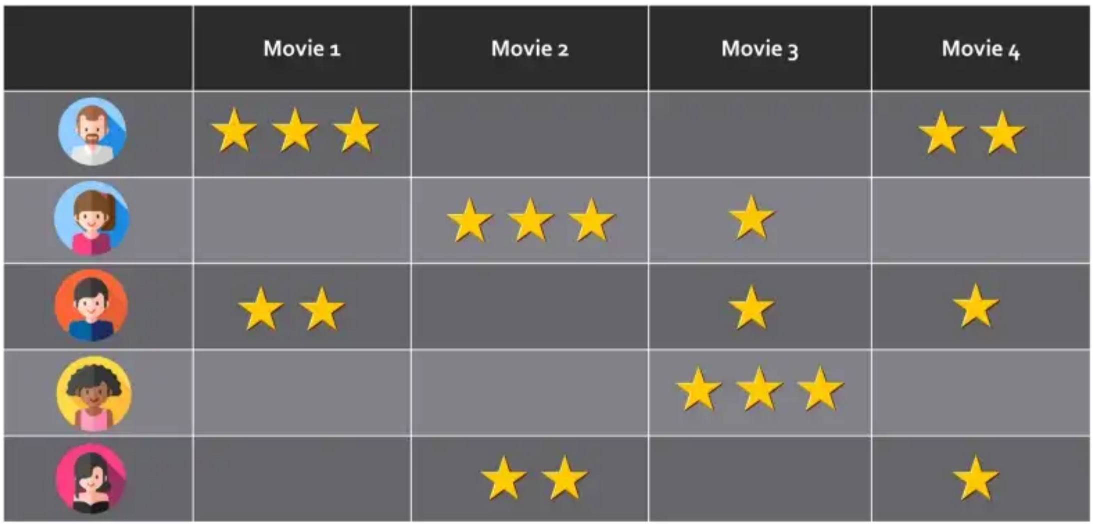
Content-based filtering: These make use of user preferences in order to make new predictions. When a user provides explicit information about his/her preferences, this is recorded and used by the system to automatically make suggestions. Many of the websites and social media that we commonly use everyday fall into this category.

Collaborative filtering: What happens when there isn't enough information provided by a user to make item recommendations? Well, in these cases we can use data provided by other users with similar preferences. Methods within this category make use of the history of past choices of a group of users in order to elicit recommendations.

Whereas in the first case it is expected for a given user to build a profile that clearly states preferences, in the second scenario this information may not be fully available, but we expect our system to still be able to make recommendations based on evidence that similar users provide. A method, known as *Probabilistic Matrix Factorization*, or *PMF* for short, is commonly used for collaborative filtering, and will be the topic of our discussion for the remainder of this article. Let us now delve into the details of this algorithm as well as its intuitions.

Probabilistic Matrix Factoriza

Let's suppose we have a set of users $u1$, $u2$, $u3$ … uN who rate a set of items $v1$, $v2$, $v3$ … vM. We can then structure the ratings as a matrix ***R*** of *N* rows and *M* columns, where *N* is the number of users and *M* is the number of items to rate.



	Movie 1	Movie 2	Movie 3	Movie 4
	★★★			★★
		★★★	★	
	★★		★	★
			★★★	
		★★		★

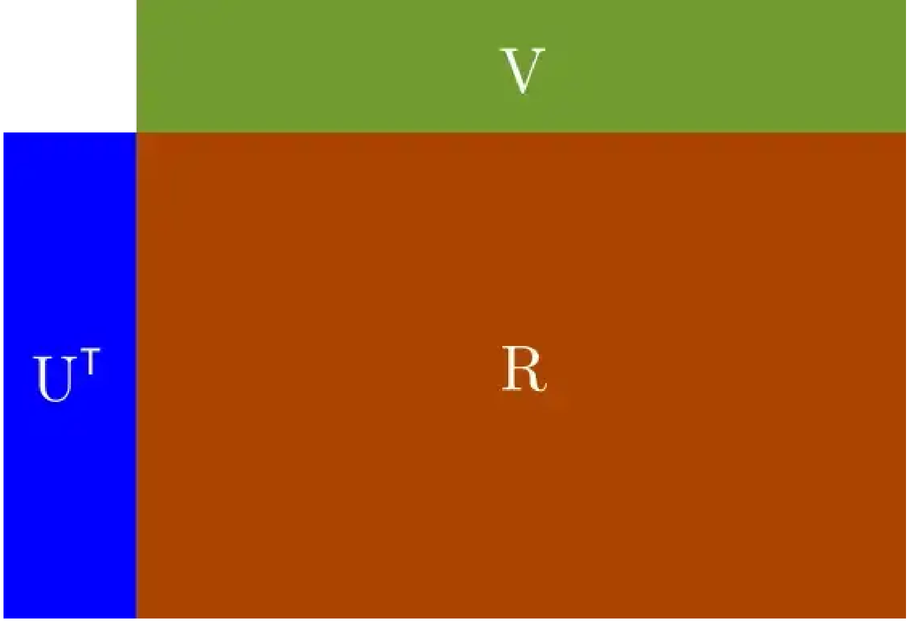
Ratings mapping. It can be thought of as a matrix where each user (rows) rates a number of items (columns)

One important trait of the ***R*** matrix is that it is *sparse*. That is, only some of its cells will have a non-empty rating value, while others will not. For a given user *A*, the system should be able to offer item recommendations based on his/her preferences as well as the choices made by similar users. However, it is not necessary for user *A* to have explicitly rated a particular item for it to be recommended. Other users with similar preferences will make up for the missing data about user *A*. This is why *Probabilistic Matrix Factorization* falls into the category of *collaborative filtering* recommender systems.

Let's think for a moment about a recommender system for movies. Imagine what things would be like if we were required to watch and rate every single movie that shows during a particular season. That would be pretty impractical, wouldn't it? We simply lack the time to do so.

Given that not all users are able to rate all items available, we must find a way to fill in the information gaps of the ***R*** matrix and still be able to offer relevant recommendations. PMF tackles this problem by making use of ratings provided by similar users. Technically speaking, it makes use of some principles of Bayesian learning that are also applicable in other scenarios where we have scarce or incomplete data.

The ***R*** matrix can be estimated by using two low-rank matrices ***U*** and ***V*** as shown below:



Components of R matrix

Here, *U*T is an *N*x*D* matrix where *N* is the number of registered users, and *D* is the rank. *V* is a *D*x*M* matrix, where *M* is the number of items to rate. Thus the *N*x*M* ratings matrix ***R*** can be approximated by means of:

$$R = U^\top V$$

Equation 1: R expression

Our job from now on is to find ***U*T** and ***V*** which in turn, will become the parameters of our model. Because ***U*** and ***V*** are low-rank matrices, PMF is also known as a *low-rank matrix factorization problem*. Furthermore, this particular trait of the ***U*** and ***V*** matrices makes PMF scalable even for datasets containing millions of records.

PMF takes its intuitions from Bayesian learning for parameter estimation. In general, we can say that in Bayesian inference, our aim is to find a posterior distribution of the model parameters by resorting to Bayes rule:

$$p(\theta|\mathbf{X},\alpha) = \frac{p(\mathbf{X}|\theta,\alpha)p(\theta|\alpha)}{p(\mathbf{X}|\alpha)} \propto p(\mathbf{X}|\theta,\alpha)p(\theta|\alpha)$$

Equation 2: Bayes rule for inference of parameters

Here, **X** is our dataset, θ is the parameter or parameter set of the distribution. α is the hyperparameter of the distribution. p(θ|**X**,α) is the *posterior distribution*, also known as *a-posteriori*. p(**X**|θ,α) is the *likelihood*, and p(θ|α) is the *prior*. The whole idea of the training process is that as we get more information about the data distribution, we will adjust the model parameter θ to fit the data. Technically speaking, the parameters of the posterior distribution will be plugged into the prior distribution for the next iteration of the training process. That is, the posterior distribution of a given training step will eventually become the prior of the next step. This procedure will be repeated until there is little variation in the posterior distribution p(θ|**X**,α) between steps.

Now let's go back to our intuitions for PMF. As we stated earlier, our model parameters will be ***U*** and ***V****,* whereas ***R*** will be our dataset. Once trained, we will end up with a revised ***R**** matrix that will also contain ratings for user-item cells that were originally empty in ***R***. We will use this revised ratings matrix to make predictions. With these considerations, we will have:

$$\begin{aligned}\theta &= \{U, V\}\\ X &= R\\ \alpha &= \sigma^2\end{aligned}$$

Where σ is the standard deviation of a zero-mean spherical Gaussian distribution. Then, by replacing these expressions in equation 2 we will get:

$$p(U, V|R, \sigma^2) = p(R|U, V, \sigma^2)p(U, V|\sigma_U^2, \sigma_V^2)$$

Since the ***U*** and ***V*** matrices are independent of each other (users and items occur independently), then this expression can also be written like this:

$$p(U, V|R, \sigma^2) = p(R|U, V, \sigma^2)p(U|\sigma_U^2)p(V|\sigma_V^2)$$

Equation 3: A-Posteriori distribution for PMF

Now it's time to find out what each of the components of this equation will equate to. First, the likelihood function is given by:

$$p(R|U, V, \sigma^2) = \prod_{i=1}^{N}\prod_{j=1}^{M}\left[\mathcal{N}(R_{ij}|U_i^T V_j, \sigma^2)\right]^{I_{ij}}$$

Equation 4: Distribution over observed ratings [1]

Here, *I{ij}* is an indicator that will take the value of 1 when the rating at row *i* and column *j* exists, and zero otherwise. As we can see, this distribution is a spherical Gaussian with the following parameters:

$$\begin{aligned}&\text{Mean:}\quad U_i^\top V_j\\ &\text{Variance:}\quad \sigma^2\end{aligned}$$

In turn, the prior distributions for ***U*** and ***V*** are given by:

$$p(U|\sigma_U^2) = \prod_{i=1}^{N} \mathcal{N}(U_i|0, \sigma_U^2), \quad p(V|\sigma_V^2) = \prod_{j=1}^{M} \mathcal{N}(V_j|0, \sigma_V^2)$$

Equations 5 and 6: Prior distributions for U and V.

Which are two zero-mean spherical Gaussians. Then, by replacing 4, 5, and 6 in 3 and we will get:

$$p(U, V|R, \sigma^2) = \prod_{i=1}^{N} \prod_{j=1}^{M} \left[\mathcal{N}(R_{ij}|U_i^\top V_j, \sigma^2)\right]^{I_{ij}} \prod_{i=1}^{N} \mathcal{N}(U_i|0, \sigma_U^2) \prod_{j=1}^{M} \mathcal{N}(V_j|0, \sigma_V^2)$$

In order to train our model, we will seek to maximize this function with respect to the parameters ***U*** and ***V*** by equating their derivatives to zero. However, doing so will be very difficult because of the *exp* functions in the Gaussians. To overcome this issue, we should instead apply logarithms to both sides of the previous equations and then apply the required derivatives. Therefore, by applying logarithms to both sides of the previous equation, we will get:

$$\ln p(U, V|R, \sigma^2) = \sum_{i=1}^{N} \sum_{j=1}^{M} I_{ij} \ln \left[\mathcal{N}(R_{ij}|U_i^\top V_j, \sigma^2)\right] + \sum_{i=1}^{N} \ln \mathcal{N}(U_i|0, \sigma_U^2) + \sum_{j=1}^{M} \ln \mathcal{N}(V_j|0, \sigma_V^2)$$

Which is a much easier expression to differentiate. We also know that by definition the Gaussian PDF is given by:

$$\mathcal{N}(X|\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(X-\mu)^2}{2\sigma^2}\right)$$

Therefore, our expression for the log-posterior will look like this (Note: For simplicity, we have gotten rid of the constants):

$$\ln p(U, V|R, \sigma^2) = -\frac{1}{2\sigma^2} \sum_{i=1}^{N} \sum_{j=1}^{M} I_{ij} (R_{ij} - U_i^\top V_j)^2 - \frac{1}{2\sigma_U^2} \sum_{i=1}^{N} \|U_i\|_{Fro}^2 - \frac{1}{2\sigma_V^2} \sum_{j=1}^{M} \|V_j\|_{Fro}^2$$

Here, the *Fro* suffix denotes the Frobenius norm which is given by:

$$\|X\|_{Fro}^2 = X^\top X$$

Finally, by introducing some additional notation to identify the hyperparameters of the model, we will get:

$$L = -\frac{1}{2}\left(\sum_{i=1}^{N}\sum_{j=1}^{M}(R_{ij} - U_i^T V_j)^2_{(i,j)\in\Omega_{R_{ij}}} + \lambda_U \sum_{i=1}^{N} \|U_i\|^2_{Fro} + \lambda_V \sum_{j=1}^{M} \|V_j\|^2_{Fro}\right)$$

Equation 7: Log-a posteriori for PMF

Where:

$$\lambda_U = \frac{\sigma_U^2}{\sigma^2}, \quad \lambda_V = \frac{\sigma_V^2}{\sigma^2}$$

Then, by differentiating equation 7 with respect to the parameters and equating the derivatives to zero, we will get:

$$\nabla_{U_i} L = \left[\sum_{j=1}^{M}(R_{ij} - U_i^\top V_j)V_j^\top\right]_{(i,j)\in\Omega_{R_{ij}}} - \lambda_U U_i = 0$$
$$\nabla_{V_j} L = \left[\sum_{i=1}^{N}(R_{ij} - U_i^\top V_j)U_i^\top\right]_{(i,j)\in\Omega_{R_{ij}}} - \lambda_V V_j = 0$$

From here, we can derive the expressions to update ***Ui*** and ***Vj***:

$$U_i = \left[\left(V_j V_j^\top\right)_{j\in\Omega_{U_i}} + \lambda_U I\right]^{-1} \left(R_{ij} V_j^\top\right)_{j\in\Omega_{U_i}}$$
$$V_j = \left[\left(U_i U_i^\top\right)_{i\in\Omega_{V_j}} + \lambda_V I\right]^{-1} \left(R_{ij} U_i^\top\right)_{i\in\Omega_{V_j}}$$

Equations 8 and 9: Expressions for updating U and V

Provided that *λU* and *λV* are both non-zero, the involved inverse matrices are guaranteed to exist. As part of the training process, we will iteratively update both ***Ui*** and ***Vj***. Once we find the optimal values for these, we will be able to obtain the value for the log-MAP (Maximum a posteriori) by using equation 7. As we will see in the Python implementation, we can use this value to monitor training convergence.

Implementation in Python

Note: The full source code for the implementation is available at https://bit.ly/35Cr5kl

For training purposes, we have made use of a subset of the IMDB movie database, and then separated it in two parts for training and validation, respectively.

Initialization: In order to initialize *V*, we draw random numbers from a zero-mean Gaussian with standard deviation 1/*λV*. Furthermore, the rank value *D* is set to a relatively small value of 10.

```python
def initialize_parameters(lambda_U, lambda_V):
    U = np.zeros((n_dims, n_users), dtype=np.float64)
    V = np.random.normal(0.0, 1.0 / lambda_V, (n_dims, n_movies))

    parameters['U'] = U
    parameters['V'] = V
    parameters['lambda_U'] = lambda_U
    parameters['lambda_V'] = lambda_V
```

initialize_pmf.py hosted with ❤ by GitHub view raw

Initialization code

Updating parameters: For updating *U* and *V*, we use equations 8 and 9:

$$U_i = \left[\left(V_j V_j^\top\right)_{j\in\Omega_{U_i}} + \lambda_U I\right]^{-1} \left(R_{ij} V_j^\top\right)_{j\in\Omega_{U_i}}$$

$$V_j = \left[\left(U_i U_i^\top\right)_{i\in\Omega_{V_j}} + \lambda_V I\right]^{-1} \left(R_{ij} U_i^\top\right)_{i\in\Omega_{V_j}}$$

And the corresponding Python code is the following:

```python
def update_parameters():
    U = parameters['U']
    V = parameters['V']
    lambda_U = parameters['lambda_U']
    lambda_V = parameters['lambda_V']

    for i in range(n_users):
        V_j = V[:, R[i, :] > 0]
        U[:, i] = np.dot(np.linalg.inv(np.dot(V_j, V_j.T) + lambda_U * np.identity(n_dims)), np.

    for j in range(n_movies):
        U_i = U[:, R[:, j] > 0]
        V[:, j] = np.dot(np.linalg.inv(np.dot(U_i, U_i.T) + lambda_V * np.identity(n_dims)), np.

    parameters['U'] = U
    parameters['U'] = U
    parameters['V'] = V
```

$$L = -\frac{1}{2}\left(\sum_{i=1}^{N}\sum_{j=1}^{M}(R_{ij} - U_i^T V_j)^2_{(i,j)\in\Omega_{R_{ij}}} + \lambda_U \sum_{i=1}^{N} \|U_i\|^2_{Fro} + \lambda_V \sum_{j=1}^{M} \|V_j\|^2_{Fro}\right)$$

With the following Python code:

```python
def log_a_posteriori():
    lambda_U = parameters['lambda_U']
    lambda_V = parameters['lambda_V']
    U = parameters['U']
    V = parameters['V']

    UV = np.dot(U.T, V)
    R_UV = (R[R > 0] - UV[R > 0])

    return -0.5 * (np.sum(np.dot(R_UV, R_UV.T)) + lambda_U * np.sum(np.dot(U, U.T)) + lambda_V *
```

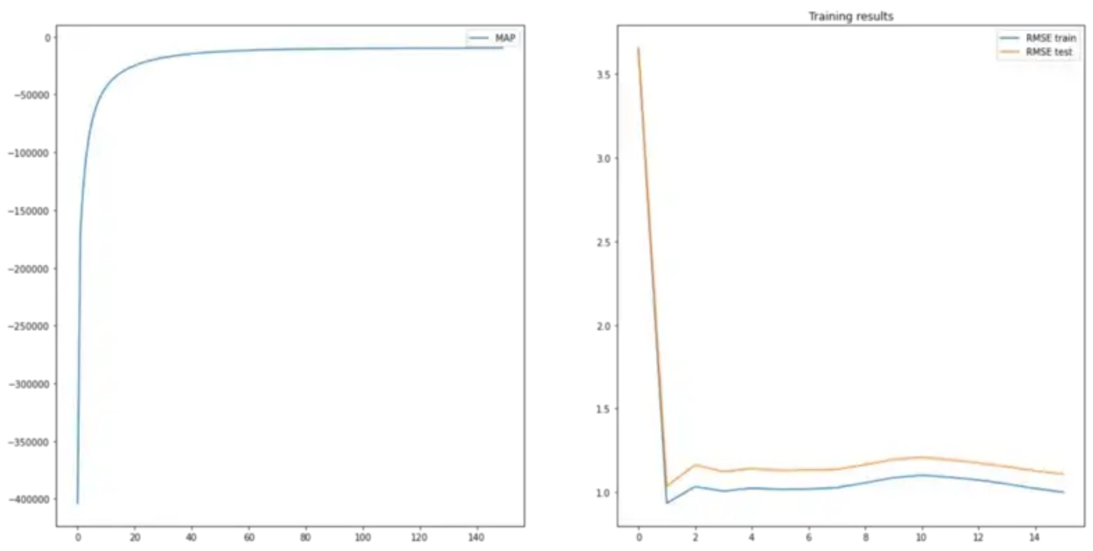
Log-posterior calculation

Training loop: To train the model we call the previous functions and monitor the log-posterior as well as the RMSE (Root Mean Square Error) evaluated on the training and testing sets:

```python
def train(n_epochs):
    initialize_parameters(0.3, 0.3)
    log_aps = []
    rmse_train = []
    rmse_test = []

    update_max_min_ratings()
    rmse_train.append(evaluate(train_set))
    rmse_test.append(evaluate(test_set))

    for k in range(n_epochs):
        update_parameters()
        log_ap = log_a_posteriori()
        log_aps.append(log_ap)

        if (k + 1) % 10 == 0:
            update_max_min_ratings()

            rmse_train.append(evaluate(train_set))
            rmse_test.append(evaluate(test_set))
            print('Log p a-posteriori at iteration', k + 1, ':', log_ap)

    update_max_min_ratings()

    return log_aps, rmse_train, rmse_test
```

Training loop

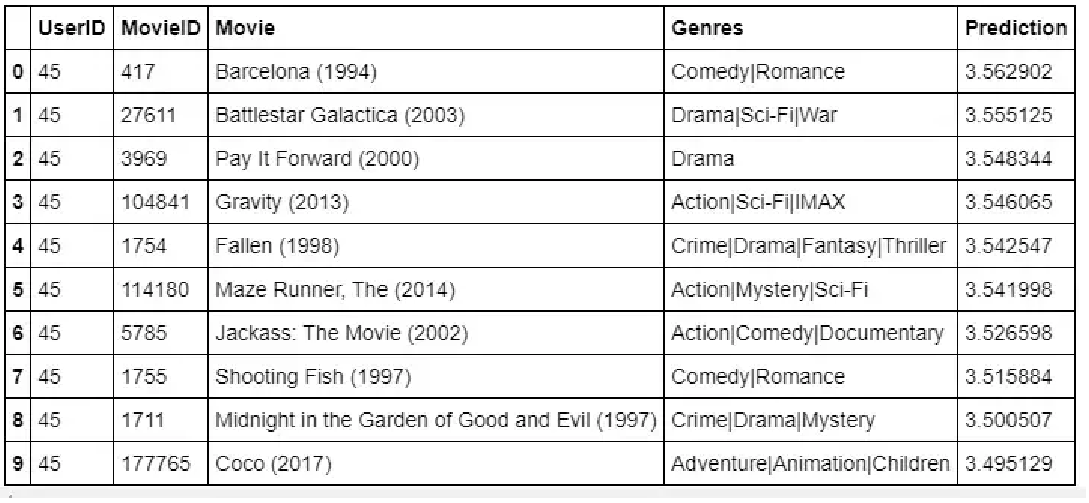
We run the training loop for 150 iterations, with the following results:



Training graphs

On the left, we can see how the log-posterior evolves as we train the model. On the right we can see the RMSE values evaluated on both the training and testing sets. Considering the ***R*** predictions may be outside of the 0–5 range for ratings, we use a linear interpolation to ensure the ***R*** values are bounded by this interval. The original paper [1] suggests other approaches, such as using a logistic function and linear interpolation. For training, gradient descent with momentum is also suggested to deal with larger datasets.

Finally, here are some movie recommendations for user id 45 in the database:



	UserID	MovieID	Movie	Genres	Prediction
0	45	417	Barcelona (1994)	Comedy|Romance	3.562902
1	45	27611	Battlestar Galactica (2003)	Drama|Sci-Fi|War	3.555125
2	45	3969	Pay It Forward (2000)	Drama	3.548344
3	45	104841	Gravity (2013)	Action|Sci-Fi|IMAX	3.546065
4	45	1754	Fallen (1998)	Crime|Drama|Fantasy|Thriller	3.542547
5	45	114180	Maze Runner, The (2014)	Action|Mystery|Sci-Fi	3.541998
6	45	5785	Jackass: The Movie (2002)	Action|Comedy|Documentary	3.526598
7	45	1755	Shooting Fish (1997)	Comedy|Romance	3.515884
8	45	1711	Midnight in the Garden of Good and Evil (1997)	Crime|Drama|Mystery	3.500507
9	45	177765	Coco (2017)	Adventure|Animation|Children	3.495129

Conclusion

PMF is a powerful algorithm for collaborative filtering. It makes use of data provided by users with similar preferences to offer recommendations to a particular user. It is also known as a low-rank matrix factorization method because it uses low rank matrices to estimate the ratings ***R*** matrix, and then make useful predictions.

I hope you enjoyed this article! Feel free to approach me in case you have any questions. I'll keep writing more material like this soon. Stay tuned.

References

[1] Salakhutdinov, Ruslan & Mnih, Andriy. *Probabilistic Matrix Factorization*. In NIPS'07: Proceedings of the 20th International Conference on Neural Information Processing Systems, pages 1257–1264, 2007.

[2] Brooks-Bartlett Jonny. *Probability concepts explained: Bayesian inference for parameter estimation*. Available at: https://bit.ly/3bajPNC

Machine Learning Pmf Bayesian Recommender Systems Towards Data Science