Bayesian Probabilistic Matrix Factorization

Ruixuan Xu
Department of Computer Science and Engineering
The Chinese University of Hong Kong
1155211213@link.cuhk.edu.hk

Xiangxiang Weng
Department of Computer Science and Engineering
The Chinese University of Hong Kong
1155211173@link.cuhk.edu.hk

Abstract

Matrix factorization is a widely used technique in recommendation systems. Probabilistic Matrix Factorization (PMF) [1] extends traditional matrix factorization by incorporating probability distributions over latent factors, allowing for uncertainty quantification. However, computing the posterior distribution is intractable due to the high-dimensional integral. To address this, we employ two Bayesian inference methods: Markov Chain Monte Carlo (MCMC) [2] and Variational Inference (VI) [3] to approximate the posterior. We evaluate their performance on MovieLens dataset and compare their convergence speed, predictive accuracy, and computational efficiency. Experimental results demonstrate that VI offers faster convergence, while MCMC provides more accurate posterior estimates.

1 Introduction

Collaborative filtering is an essential technique in recommendation systems, where the goal is to predict user preferences based on sparse observed ratings. Matrix factorization has been widely adopted due to its ability to model user-item interactions effectively. However, traditional matrix factorization relies on point estimates, which may lead to overfitting and lack of uncertainty quantification.

Probabilistic Matrix Factorization (PMF) [1] mitigates this issue by considering a probabilistic approach, where user and item latent matrices are treated as random variables with prior distributions. The challenge in PMF is computing the posterior distribution of latent matrices, which is intractable. To approximate the posterior, we explore two Bayesian inference methods:

1. Markov Chain Monte Carlo (MCMC) [2]: A sampling-based approach that provides asymptotically exact posterior estimates.

2. Variational Inference (VI) [3]: An optimization-based approach that approximates the posterior using a parameterized distribution.

In this report, we implement both methods on MovieLens dataset and compare their performance.

2 Problem Setting

2.1 Matrix Factorization Model

In mathematics, a sparse matrix refers to a matrix in which most of the elements are zero. A sparse rating matrix is a common data structure in recommendation systems, typically used to represent

users' rating data for items. Collaborative filtering relies on user behavior data such as ratings to predict items a user might like. Due to the high sparsity of rating matrices, directly storing and computing them would waste a lot of memory and computational resources. Matrix factorization addresses this by decomposing the user-item rating matrix into two low-dimensional matrices, and then predicting ratings via their dot product, thereby achieving collaborative filtering.

Given an $N \times M$ sparse rating matrix R:

Figure 1: Sparse Rating Matrix

Suppose there are L observed ratings ($L \ll N \times M$), for predicting all unrated items, we need to estimate $N \times M - L \approx N \times M$ parameters. An $N \times M$ rating matrix R can be approximated by the multiplication of an $N \times K$ matrix and a $K \times M$ matrix.

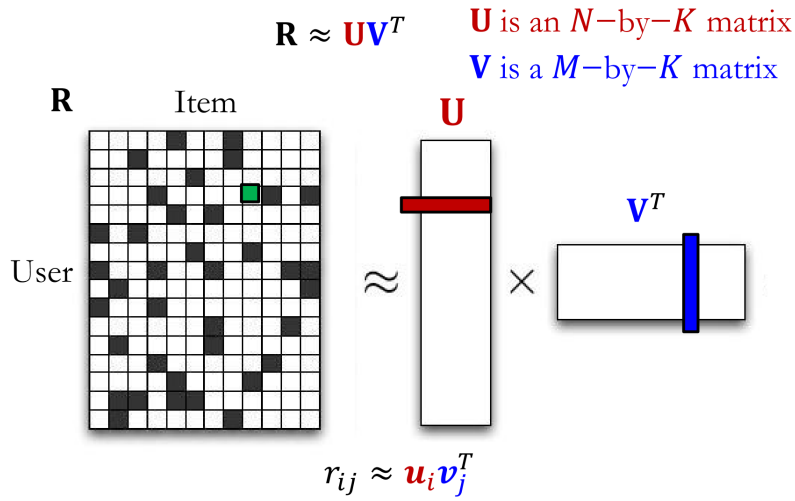

Figure 2: Matrix Factorization

Here, K is feature dimension, $U \in \mathbb{R}^{N \times K}$ is user feature matrix, and $V \in \mathbb{R}^{M \times K}$ is item feature matrix. With matrix factorization, rather than estimating $N \times M - L \approx N \times M$ parameters, we only need to estimate $(N + M) \times K$, where $K \ll \min(N, M)$. Our goal is to find matrices U and V such that UV^T can accurately represent the original rating matrix. In this way, the number of parameters to be estimated is greatly reduced, and the entire prediction matrix can be obtained simply by matrix multiplication, making computation efficient. Finding the target parameter matrices U and V involves optimization methods based on gradient descent. For example, minimizing the following loss function:

$$\min_{U,V} \sum_{(i,j) \in O} (r_{ij} - \mathbf{u}_i \mathbf{v}_j^T)^2$$

where:

- O is the set of all rated items.

- r_{ij} is the true rating given by user i to item j.

- $\mathbf{u}_i \mathbf{v}_j^T$ is the predicted rating.

Gradient descent update rules:

$$\mathbf{u}_i \leftarrow \mathbf{u}_i + \alpha \cdot \sum_{j \in O_i} (r_{ij} - \mathbf{u}_i \mathbf{v}_j^T) \mathbf{v}_j$$
$$\mathbf{v}_j \leftarrow \mathbf{v}_j + \alpha \cdot \sum_{i \in O_j} (r_{ij} - \mathbf{u}_i \mathbf{v}_j^T) \mathbf{u}_i$$

where α is the learning rate.

However, traditional matrix factorization has many limitations, such as poor generalization ability, inability to capture the uncertainty of latent vectors and lack of probabilistic interpretation. Therefore, we introduce Bayesian Probabilistic Matrix Factorization (BPMF) on the basis of matrix factorization.

2.2 Bayesian Probabilistic Matrix Factorization

Instead of estimating matrices U and V to compute each rating r_{ij} based on deterministic approaches, one can use Bayesian inference. Consider each row of U and V, i.e., $\mathbf{U}_i$ and $\mathbf{V}_j$, as a multivariate random variables. $\mathbf{U}_i$ and $\mathbf{V}_j$ are assumed to be standard normal random vectors.

$$\begin{cases} U_{ik} \sim N(0,1) & \text{for any } 1 \leq k \leq K \\ V_{ik} \sim N(0,1) & \text{for any } 1 \leq k \leq K \end{cases}$$

The prior distributions of $\mathbf{U}_i$ and $\mathbf{V}_j$ have the following PDFs

$$\begin{cases} f_{\mathbf{U}_i}(\mathbf{u}_i) = \frac{1}{(2\pi)^{\frac{K}{2}}} \exp\left(-\frac{1}{2} \mathbf{u}_i \mathbf{u}_i^T\right) \\ f_{\mathbf{V}_j}(\mathbf{v}_j) = \frac{1}{(2\pi)^{\frac{K}{2}}} \exp\left(-\frac{1}{2} \mathbf{v}_j \mathbf{v}_j^T\right) \end{cases}$$

The likelihood of each observed rating r_{ij} is defined as a normal distribution:

$$R_{ij} | \mathbf{U}_i, \mathbf{V}_j \sim N\left(\text{sigmoid}\left(\mathbf{u}_i \mathbf{v}_j^T\right), \sigma^2\right)$$

Note: as sigmoid function returns a value into $[0, 1]$, during training, we need to first normalize ratings via $r_{ij} \leftarrow \frac{r_{ij}-1}{R-1}$, where original ratings are defined in $\{1, 2, \ldots, R\}$, and σ^2 is a hyperparameter.

The PDF of likelihood is

$$f_{R_{ij}|\mathbf{U}_i, \mathbf{V}_j}(r_{ij} | \mathbf{u}_i, \mathbf{v}_j) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(r_{ij} - \text{sigmoid}\left(\mathbf{u}_i \mathbf{v}_j^T\right)\right)^2}{2\sigma^2}\right).$$

Based on conditional independence and Bayes' rule, the posterior of $\mathbf{U}$ and $\mathbf{V}$ can be inferred as

$$\begin{aligned} & f_{\mathbf{U},\mathbf{V}|\{\mathbf{R}_{ij}\}}(U, V | \{r_{ij}\}) \\ \propto\ & f_{\{\mathbf{R}_{ij}\}|\mathbf{U},\mathbf{V}}(\{r_{ij}\} | U, V) f_{\mathbf{U},\mathbf{V}}(U, V) \\ \propto\ & \prod_{i=1}^{N} \prod_{j=1}^{M} \left[f_{R_{ij}|\mathbf{U}_i, \mathbf{V}_j}(r_{ij} | \mathbf{u}_i, \mathbf{v}_j)\right]^{I_{ij}} \exp\left(-\frac{1}{2} \mathbf{u}_i \mathbf{u}_i^T\right) \exp\left(-\frac{1}{2} \mathbf{v}_j \mathbf{v}_j^T\right). \end{aligned}$$

where I_{ij} is an indicator function that is equal to 1 if user i rated movie j, otherwise 0.

If the likelihood adopts a simple Gaussian distribution, minimizing the negative log of the posterior (loss function) combined with gradient descent to obtain the most likely U and V, and then directly multiplying them to generate the prediction matrix are still available, just as researchers do in PMF [1]. However, we introduce nonlinearity by applying a sigmoid function to the mean of the Gaussian likelihood distribution, in order to simulate the complex likelihoods that are likely to occur in real-world applications. In addition, a simple Gaussian likelihood may produce predicted ratings outside the valid range (e.g., from 1 to 5), while sigmoid can compress the predicted ratings into valid range.

In this case, the gradient descent becomes infeasible. Therefore, we adopt Bayesian framework, retain posterior distributions of U and V, and optimizing the predictive distribution over unrated data.

To make a prediction on an unobserved rating r_{ab}:

$$\begin{aligned} & f_{R_{ab}|\{R_{ij}\}}(r_{ab}|\{r_{ij}\}) \\ = & \int_{\mathbf{u}_a \in \mathbb{R}^K} \int_{\mathbf{v}_b \in \mathbb{R}^K} f_{R_{ab}|\mathbf{U}_a,\mathbf{V}_b}(r_{ab}|\mathbf{u}_a,\mathbf{v}_b) f_{\mathbf{U}_a,\mathbf{V}_b|\{R_{ij}\}}(\mathbf{u}_a,\mathbf{v}_b|\{r_{ij}\}) d\mathbf{u}_a d\mathbf{v}_b \\ = & \mathbb{E}_{\mathbf{U}_a,\mathbf{V}_b|\{R_{ij}\}} \left[f_{R_{ab}|\mathbf{U}_a,\mathbf{V}_b}(r_{ab}|\mathbf{u}_a,\mathbf{v}_b) \right]. \end{aligned}$$

In general, we can use MAP to generate a predicted rating in $[0, 1]$:

$$r_{ab}^* = \arg\max_{r_{ab}} f_{R_{ab}|\{R_{ij}\}}(r_{ab}|\{r_{ij}\}).$$

Then map back to the original rating using $r = (R-1)r^* + 1$.

However, we still cannot compute $f_{R_{ab}|\{R_{ij}\}}(r_{ab}|\{r_{ij}\})$, because there are two challenges:

1. Although we have derived the parameter posterior distribution $f_{\mathbf{U}_a,\mathbf{V}_b|\{R_{ij}\}}(\mathbf{u}_a,\mathbf{v}_b|\{r_{ij}\})$, we cannot accurately compute the normalization constant, since integrating a high-dimensional Gaussian distribution with a sigmoid is difficult even for computers;

2. Even if we obtain the parameter posterior, the integral involved in the predictive distribution itself is high-dimensional and difficult to compute.

To address this, we introduce some Bayesian inference methods [4], [5].

3 Bayesian Inference Methods

Consider Bayesian inference:

$$f_{\Theta|X}(\theta|x) = \frac{f_\Theta(\theta) f_{X|\Theta}(x|\theta)}{Z(x)}.$$

- $f_{\Theta|X}(\theta|x)$: Posterior.

- $f_\Theta(\theta)$: Prior.

- $f_{X|\Theta}(x|\theta)$: Likelihood.

- $Z(x)$: Normalization constant, where $Z(x) = \int_\theta f_\Theta(\theta) f_{X|\Theta}(x|\theta) d\theta$.

For multiple observations, let $D = \{X_1, \ldots, X_n\}$ be the joint random variables and $d = \{x_1, \ldots, x_n\}$ their values. Bayesian inference can be written as:

$$f_{\Theta|D}(\theta|d) = \frac{f_\Theta(\theta) f_{D|\Theta}(d|\theta)}{Z(d)}.$$

Bayesian prediction:

$$f_{X|D}(x^*|d) = \int_{-\infty}^{+\infty} f_{X|\Theta}(x^*|\theta) f_{\Theta|D}(\theta|d) d\theta = \mathbb{E}_{\Theta|D=d}[f_{X|\Theta}(x^*|\theta)].$$

Two major computational challenges:

1. Computing $Z(d)$ requires computing high dimensional integrals as θ is multivariate in practice;

2. The integral involved in the predictive distribution itself is difficult to compute.

We propose two solutions:

1. Use MCMC [2] to sample from $f_{\Theta|D}(\theta|d)$ and use the sample mean to approximate the expectation.

2. Use VI [3] to approximate $f_{\Theta|D}(\theta|d)$ by $q(\theta)$, which is easy to integrate.

3.1 Markov Chain Monte Carlo (MCMC)

Suppose $\mathbf{z}$ is multi-variate random variable, and we are interested in evaluating the expectation:

$$\mathbb{E}_{\mathbf{Z}}\left[h(\mathbf{z})\right] = \int_{\mathbf{z}} h(\mathbf{z}) f(\mathbf{z}) d\mathbf{z}.$$

Where

$$f(\mathbf{z}) = \frac{g(\mathbf{z})}{Z}.$$

Our objective is to draw independent samples $\{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$ from $f(\mathbf{z})$ to approximate $\mathbb{E}_{\mathbf{Z}}\left[h(\mathbf{z})\right]$:

$$\mathbb{E}_{\mathbf{Z}}\left[h(\mathbf{z})\right] \approx \frac{1}{n} \sum_{i=1}^{n} h(\mathbf{z}_i).$$

In high-dimensional settings, Markov Chain Monte Carlo (MCMC) is widely used for sampling. The Metropolis-Hastings algorithm is a commonly used MCMC method.

Core idea: Construct a proposal distribution $q(\mathbf{z}'|\mathbf{z})$ to generate candidate samples and use an acceptance rate to decide whether to accept the sample.

Detailed steps: Let the target distribution be $f(\mathbf{z}) = \frac{g(\mathbf{z})}{Z}$.

1. Construct a proposal distribution $q(\mathbf{z}'|\mathbf{z})$.

2. Choose an initial state $\mathbf{z}_0$. Set the initial time $t = 0$.

3. Sample a candidate $\mathbf{z}'$ from $q(\mathbf{z}'|\mathbf{z}_t)$.

4. Compute the acceptance rate:

$$\alpha(\mathbf{z}_t, \mathbf{z}') = \min\left(1, \frac{g(\mathbf{z}')q(\mathbf{z}_t|\mathbf{z}')}{g(\mathbf{z}_t)q(\mathbf{z}'|\mathbf{z}_t)}\right).$$

5. Accept the new sample with probability α. If accepted, set $\mathbf{z}_{t+1} = \mathbf{z}'$; otherwise, set $\mathbf{z}_{t+1} = \mathbf{z}_t$.

6. Update time $t \leftarrow t + 1$.

7. Repeat steps 3–6 until the samples meet the requirements.

The specific iterative process is as follows:

Algorithm 1 Metropolis-Hastings Algorithm for MCMC

Input: Unnormalized target density $g(\mathbf{z})$, proposal distribution $q(\mathbf{z}'|\mathbf{z})$
Output: Samples $\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n\}$ approximating $f(\mathbf{z})$
Initialize: Initial state $\mathbf{z}_0$, set $t = 0$
while $t < n$ **do**
    Sample a candidate $\mathbf{z}' \sim q(\mathbf{z}'|\mathbf{z}_t)$
    Compute acceptance rate:

$$\alpha(\mathbf{z}_t, \mathbf{z}') = \min\left(1, \frac{g(\mathbf{z}')q(\mathbf{z}_t|\mathbf{z}')}{g(\mathbf{z}_t)q(\mathbf{z}'|\mathbf{z}_t)}\right)$$

    Sample $u \sim \text{Uniform}(0, 1)$
    if $u < \alpha(\mathbf{z}_t, \mathbf{z}')$ **then**
        Accept: set $\mathbf{z}_{t+1} = \mathbf{z}'$
    else
        Reject: set $\mathbf{z}_{t+1} = \mathbf{z}_t$
    end if
    Update $t \leftarrow t + 1$
end while
return $\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n\}$

For mathmatical details, please check Appendix.

3.2 Variational Inference (VI)

Due to the intractability of the exact posterior $P(\mathbf{U}, \mathbf{V} \mid \{r_{ij}\})$, we employ Variational Inference (VI) to approximate it. We define a variational distribution $Q(\mathbf{U}, \mathbf{V})$ under the mean-field assumption:

$$Q(\mathbf{U}, \mathbf{V}) = \prod_{i=1}^{N} Q_i(\mathbf{u}_i) \prod_{j=1}^{M} Q_j(\mathbf{v}_j),$$

and optimize it to minimize the KL divergence between the true posterior and the variational distribution. This leads to the maximization of the Evidence Lower Bound (ELBO):

$$\log P(\{r_{ij}\}) \geq \mathcal{L}(Q) = \mathbb{E}_Q[\log P(\{r_{ij}\}, \mathbf{U}, \mathbf{V})] - \mathbb{E}_Q[\log Q(\mathbf{U}, \mathbf{V})].$$

We derive the update rules using Coordinate Ascent Variational Inference (CAVI), iteratively optimizing each variational factor while keeping others fixed.

The specific iterative process is as follows:

Algorithm 2 Coordinate Ascent Variational Inference (CAVI)

Input: A model $p(\mathbf{x}, \mathbf{z})$, a data set $\mathbf{x}$
Output: A variational density $Q(\mathbf{z}) = \prod_{j=1}^{m} Q_j(z_j)$
Initialize: Variational factors $Q_j(z_j)$
while the ELBO has not converged **do**
  for $j \in \{1, \ldots, m\}$ **do**
    Set $Q_j(z_j) \propto \exp\{\mathbb{E}_{-j}[\log p(z_j \mid \mathbf{z}_{-j}, \mathbf{x})]\}$
  end for
  Compute $\text{ELBO}(Q) = \mathbb{E}[\log p(\mathbf{z}, \mathbf{x})] - \mathbb{E}[\log Q(\mathbf{z})]$
end while
return $q(\mathbf{z})$

For mathmatical details, please check Appendix.

4 Dataset Processing

We use the MovieLens-small dataset, which consists of 100,836 ratings from 610 users on 9,724 movies. The rating data is stored in the `ratings.csv` file. We implemented two Python scripts: `MCMC.py` and `VI.py`, which perform Bayesian matrix completion using the MCMC and VI methods, respectively, on the data from the CSV file.

The data in `ratings.csv` is stored in the following format:

userId	movieId	rating	timestamp
1	1	4	964982703
1	3	4	964981247
1	6	4	964982224
1	47	5	964983815
1	50	5	964982931
⋮	⋮	⋮	⋮

We ignore the `timestamp` column and import the first three columns into the Python scripts for further processing. The dataset is preprocessed by:

1. Normalizing ratings between 0 and 1.

2. Divided into three groups: 60% training set, 20% validation set, and 20% test set.

5 Experimental Evaluation

We evaluate MCMC and VI on the MovieLens dataset based on:

1. Convergence Speed;

2. Predictive Accuracy;

3. Computational Efficiency.

5.1 Convergence Speed

Controlling other parameters such as latent vector dimension and variance to be consistent, we set 300 epochs for VI and found that it generally began to stabilize between 150 and 200 epochs; we set 1000 epochs for MCMC and found that it generally began to stabilize between 600 and 700 epochs. This indicates that VI requires fewer epochs to converge compared to MCMC.

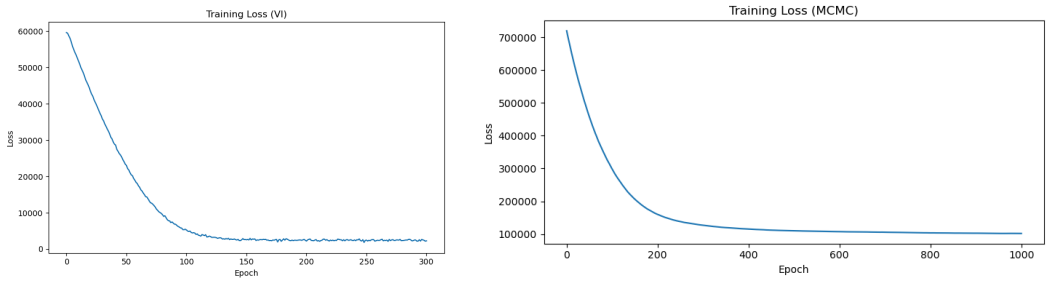

Figure 3: Loss-Epoch Plot of VI and MCMC

5.2 Predictive Accuracy

Measured using RMSE, where VI achieves 1.2277 and MCMC achieves 1.1836.

```
Epoch 294/300, Loss: 2558.8389
Epoch 295/300, Loss: 2205.0576
Epoch 296/300, Loss: 2657.1338
Epoch 297/300, Loss: 2515.6631
Epoch 298/300, Loss: 2419.4246
Epoch 299/300, Loss: 2167.2266
Epoch 300/300, Loss: 2255.2144
Test RMSE: 1.2277, Test MAE: 1.0127
```

```
Epoch 996/1000, Loss: 101890.5703
Epoch 997/1000, Loss: 101837.4375
Epoch 998/1000, Loss: 101859.8594
Epoch 999/1000, Loss: 101881.4297
Epoch 1000/1000, Loss: 101904.0000
Test RMSE: 1.1836, Test MAE: 0.8882
```

Figure 4: RMSE of VI and MCMC

5.3 Computational Efficiency

We used an NVIDIA GeForce RTX 4060 Laptop GPU for computation. The execution time of VI.py was approximately 6 seconds, while MCMC.py took about 6 hours to run. This means VI runs approximately 3,600x faster than MCMC due to its optimization-based approach.

5.4 Discussion

1. MCMC is more accurate as it samples from the true posterior.

2. VI is computationally efficient and suitable for large-scale data.

3. Trade-off: If accuracy is paramount, use MCMC; if speed is critical, use VI.

6 Conclusion

In this report, we proposed a Bayesian approach to matrix factorization, leveraging two prominent inference methods—Markov Chain Monte Carlo (MCMC) and Variational Inference (VI)—to address

the intractability of the posterior distribution over latent user and item features in collaborative filtering. We formulated the probabilistic model by introducing Gaussian priors on user and item latent vectors and a sigmoid-transformed Gaussian likelihood to ensure bounded rating predictions.

We implemented both inference techniques and evaluated their performance on the MovieLens dataset, focusing on three key dimensions: convergence speed, predictive accuracy, and computational efficiency. Our experiments demonstrated that VI converges considerably faster than MCMC and achieves remarkable computational efficiency due to its deterministic optimization framework. However, MCMC, by virtue of sampling from the true posterior, offers more accurate predictions, though at the cost of significantly higher runtime.

These results highlight a fundamental trade-off in Bayesian matrix factorization: MCMC yields higher fidelity at the expense of time, while VI offers scalability and speed with slightly reduced accuracy. Therefore, the choice of inference method should be guided by the specific constraints and requirements of the target application.

In future work, we plan to explore hybrid approaches that combine the strengths of MCMC and VI, such as initializing MCMC with variational parameters or employing amortized inference techniques. Additionally, extending the model to incorporate content-based features or temporal dynamics could further enhance recommendation accuracy and applicability in real-world systems.

References

[1] Mnih, A., & Salakhutdinov, R. R. (2007). Probabilistic matrix factorization. Advances in neural information processing systems, 20.

[2] W. K. Hastings. (1970). Monte Carlo sampling methods using Markov chains and their applications, Biometrika, Volume 57, Issue 1, April 1970, Pages 97–109, https://doi.org/10.1093/biomet/57.1.97

[3] Blei, D. M., Kucukelbir, A., & McAuliffe, J. D. (2017). Variational Inference: A Review for Statisticians. Journal of the American Statistical Association, 112(518), 859–877. https://doi.org/10.1080/01621459.2017.1285773

[4] Ruslan Salakhutdinov and Andriy Mnih. (2008). Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In Proceedings of the 25th international conference on Machine learning (ICML '08). Association for Computing Machinery, New York, NY, USA, 880–887. https://doi.org/10.1145/1390156.1390267

[5] Guangyong Chen, Fengyuan Zhu, and Pheng Ann Heng. (2018). Large-Scale Bayesian Probabilistic Matrix Factorization with Memo-Free Distributed Variational Inference. ACM Trans. Knowl. Discov. Data 12, 3, Article 31 (June 2018), 24 pages. https://doi.org/10.1145/3161886

Appendix

1. Mathmatical Details of MCMC

A set of random variables $\{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n\}$ forms a first-order Markov chain if the following conditional independence holds:

$$P(\mathbf{z}_{k+1}|\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_k) = P(\mathbf{z}_{k+1}|\mathbf{z}_k) \qquad \text{for } k \in \{1, \ldots, n-1\}.$$

This means that the current state depends only on the previous state and not on earlier states.

For continuous variables, it becomes:

$$f(\mathbf{z}_{k+1}|\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_k) = f(\mathbf{z}_{k+1}|\mathbf{z}_k) \qquad \text{for } k \in \{1, \ldots, n-1\}.$$

To generate a Markov chain, we need:

1. Define the initial state distribution $P(\mathbf{z}_1)$.

2. Construct the transition kernels:

$$T_k(\mathbf{z}_{k+1} \leftarrow \mathbf{z}_k) = P(\mathbf{z}_{k+1}|\mathbf{z}_k), \quad k \in \{1, \ldots, n-1\}.$$

For continuous variables, the transition kernel is written as:

$$T_k(\mathbf{z}_{k+1} \leftarrow \mathbf{z}_k) = f(\mathbf{z}_{k+1}|\mathbf{z}_k), \quad k \in \{1, \ldots, n-1\}.$$

A Markov chain is called homogeneous if the transition kernels are the same for all k.

The marginal probability of a specific state can be computed via product and sum rules, i.e., the law of total probability:

$$P(\mathbf{z}_{k+1}) = \sum_{\mathbf{z}_k} T(\mathbf{z}_{k+1} \leftarrow \mathbf{z}_k) P(\mathbf{z}_k),$$

or for continuous variables, expressed in terms of PDFs:

$$f(\mathbf{z}_{k+1}) = \int_{\mathbf{z}_k} T(\mathbf{z}_{k+1} \leftarrow \mathbf{z}_k) f(\mathbf{z}_k) d\mathbf{z}_k.$$

A distribution $P^*(\cdot)$ is said to be stationary or invariant with respect to a Markov chain if each step in the chain leaves $P^*(\cdot)$ invariant:

$$P^*(\mathbf{z}) = \sum_{\mathbf{z}'} T(\mathbf{z} \leftarrow \mathbf{z}') P^*(\mathbf{z}'), \qquad \forall \mathbf{z}.$$

A sufficient (but not necessary) condition for ensuring $P^*(\cdot)$ is stationary or invariant is to choose a transition kernel that satisfies the property of detailed balance, defined by:

$$T(\mathbf{z}' \leftarrow \mathbf{z}) P^*(\mathbf{z}) = T(\mathbf{z} \leftarrow \mathbf{z}') P^*(\mathbf{z}'), \qquad \forall \mathbf{z}, \mathbf{z}'.$$

If detailed balance holds, the chain is said to be reversible.

The core idea of using Markov Chain Monte Carlo (MCMC) methods for sampling:

1. Construct a Markov chain whose stationary distribution is $f(\mathbf{z})$;

2. Simulate the chain to generate samples;

3. After a sufficient number of steps, the samples approximately follow the distribution $f(\mathbf{z})$.

When the Markov chain runs long enough, even if the initial state is not sampled from $f(\mathbf{z})$, the eventually generated samples will converge to this distribution, thus allowing approximate sampling. It should be noted that, to guarantee convergence to $f(\mathbf{z})$, the Markov chain needs to be ergodic.

2. Mathmatical Details of VI

Considering the general case, by Bayes' theorem, we have

$$\log P(X) = \log P(X, Z) - \log P(Z \mid X).$$

Divide both terms inside the $\log$ on the right-hand side by $Q(Z)$. The equation remains equivalent:

$$\log P(X) = \log \frac{P(X, Z)}{Q(Z)} - \log \frac{P(Z \mid X)}{Q(Z)}.$$

Multiply both sides of the equation by $Q(Z)$ and integrate over Z, yielding:

Left-hand side:

$$\int_Z \log P(X) Q(Z) dZ = \log P(X).$$

Right-hand side:

$$\begin{aligned}
&\int_Z Q(Z) \log \frac{P(X, Z)}{Q(Z)} dZ - \int_Z Q(Z) \log \frac{P(Z \mid X)}{Q(Z)} dZ \\
&= \int_Z Q(Z) \log \frac{P(X, Z)}{Q(Z)} dZ - \int_Z Q(Z) \log P(Z \mid X) dZ \\
&= \mathcal{L}(Q) + \mathrm{KL}(Q \| P).
\end{aligned}$$

In the above, the first term is denoted as $\mathcal{L}(Q)$, and the second term (with a negative sign) represents the KL divergence, indicating the distance between the posterior distribution $P(Z \mid X)$ and the distribution $Q(Z)$. Therefore, we have:

$$\log P(X) = \mathcal{L}(Q) + \mathrm{KL}(Q \| P).$$

Assume under the mean-field theory that $Q(Z)$ is conditionally independent for all components of Z (let Z have M components), i.e.,

$$Q(Z) = \prod_{i=1}^{M} Q_i(Z_i).$$

From the previous derivation, we know:

$$\begin{aligned}
\mathcal{L}(Q) &= \int_Z Q(Z) \log \frac{P(X, Z)}{Q(Z)} dZ \\
&= \int_Z \prod_{i=1}^{M} Q_i(Z_i) \log P(X, Z) dZ - \int_Z \prod_{i=1}^{M} Q_i(Z_i) \log \prod_{i=1}^{M} Q_i(Z_i) dZ \\
&= \int_Z \prod_{i=1}^{M} Q_i(Z_i) \log P(X, Z) dZ - \int_Z \prod_{i=1}^{M} Q_i(Z_i) \sum_{i=1}^{M} \log Q_i(Z_i) dZ.
\end{aligned}$$

We will make some transformations to the first expression on the left side:

$$\begin{aligned}
&\int_Z \prod_{i=1}^{M} Q_i(Z_i) \log P(X, Z) dZ \\
&= \int Q_j(Z_j) \left[\int \prod_{i \neq j} Q_i(Z_i) \log P(X, Z) dZ_j \right] dZ_j \\
&= \int q_j(Z_j) \mathbb{E}_{q_i(Z_i), i \neq j} \left[\log P(X, Z) \right] dZ_j.
\end{aligned}$$

Now let's make some transformations to the second expression on the right side:

$$\begin{aligned}
&\int_Z \prod_{i=1}^{M} Q_i(Z_i) \sum_{i=1}^{M} \log Q_i(Z_i) dZ \\
=& \int_Z \left(\prod_{i=1}^{M} Q_i(z_i) \right) \left(\sum_{i=1}^{M} \log Q_i(z_i) \right) dZ \\
=& \int_Z \left(\prod_{i=1}^{M} Q_i(z_i) \right) \left[\log Q_1(z_1) + \log Q_2(z_2) + \cdots + \log Q_M(z_M) \right] dZ.
\end{aligned}$$

Now we attempt to isolate one of the items to discover the pattern:

$$\begin{aligned}
&\int_Z \left(\prod_{i=1}^{M} Q_i(z_i) \right) \log Q_1(z_1)\, dZ \\
=& \int_Z Q_1 Q_2 \cdots Q_M \log Q_1 \, dZ \\
=& \int_{z_1, z_2, \ldots, z_M} Q_1 Q_2 \cdots Q_M \log Q_1 \, dz_1 dz_2 \cdots dz_M \\
=& \left(\int_{z_1} Q_1 \log Q_1 \, dz_1 \right) \left(\int_{z_2} Q_2 \, dz_2 \right) \cdots \left(\int_{z_M} Q_M \, dz_M \right) \\
=& \int_{z_1} Q_1 \log Q_1 \, dz_1.
\end{aligned}$$

Back to the expression we care about

$$\begin{aligned}
&\int_Z \left(\prod_{i=1}^{M} Q_i(z_i) \right) \left[\log Q_1(z_1) + \log Q_2(z_2) + \cdots + \log Q_M(z_M) \right] dZ \\
=& \sum_{i=1}^{M} \left(\int_{z_i} Q_i(z_i) \log Q_i(z_i) \, dz_i \right) \\
=& \int_{z_j} Q_j(z_j) \log Q_j(z_j) \, dz_j + \text{Constant}.
\end{aligned}$$

We transform the expectation to another form:

$$\mathbb{E}_{Q_i(Z_i), i \neq j} \left[\log P(X, Z) \right] = \log \tilde{P}(X, Z_j).$$

Here we adopt Coordinate Ascent Variational Inference (CAVI), the idea of CAVI is that when updating $Q_j(Z_j)$, the other $Q_i(Z_i)$ for $i \neq j$ are kept fixed, thus

$$\begin{aligned}
\mathcal{L}(Q) &= \int_{z_j} Q_j(z_j) \log \frac{\hat{P}(X, z_j)}{Q_j(z_j)} \, dz_j + \text{Constant} \\
&= -\operatorname{KL}\left(Q_j \,\|\, \hat{P}(X, z_j) \right) + \text{Constant}.
\end{aligned}$$

To maximize $\mathcal{L}(\mathbf{Q})$, we need to set $Q_j(Z_j) = \tilde{P}(X, Z_j)$, that is,

$$Q_j^*(Z_j) = \exp\left(\mathbb{E}_{Q_i(Z_i), i \neq j} \left[\log P(X, Z) \right] \right).$$

To ensure that $\sum_{Z_j} Q_j^*(Z_j) = 1$, we normalize the above expression, obtaining

$$Q_j^*(Z_j) = \frac{\exp\left(\mathbb{E}_{Q_i(Z_i), i \neq j} \left[\log P(X, Z) \right] \right)}{\int \exp\left(\mathbb{E}_{Q_i(Z_i), i \neq j} \left[\log P(X, Z) \right] \right) dZ_j}.$$

In this project, Z can be regarded as [U; V], X can be regarded as $\{r_{ij}\}$

$$\begin{aligned}
P(X, Z) &= f(\mathbf{U}, \mathbf{V}, \{r_{ij}\}) \\
&= \prod_{i=1}^{N} \prod_{j=1}^{M} \left(\mathcal{N}(r_{ij} \mid \text{sigmoid}(\mathbf{u}_i^\top \mathbf{v}_j), \sigma^2) \right)^{I_{ij}} \exp\left(-\frac{1}{2} \mathbf{u}_i^\top \mathbf{u}_i \right) \exp\left(-\frac{1}{2} \mathbf{v}_j^\top \mathbf{v}_j \right).
\end{aligned}$$