# PROBABILISTIC MATRIX FACTORIZATION VIA PROBABILISTIC PROGRAMMING

FRANK WOOD AND ZIYU WANG

## Introduction

Probabilistic matrix factorization and Bayesian variants thereof came of age in the heat of the Netflix challenge problem race. For this reason we'll use movie ratings prediction language to describe the task of finding a probabilistic, approximate matrix factorization

$$\mathbf{R} \approx \mathbf{U}\mathbf{V}$$

In such language $R_{ij}$ is the rating user $i \in \{1 \dots N\}$ gave to movie $j \in \{1 \dots M\}$. We would like to find matrixes $\mathbf{U} \in \mathbb{R}^{D \times N}$ and $\mathbf{V} \in \mathbb{R}^{D \times M}$ consisting of $D$-dimensional user- and movie-specific latent features given only sparse observations of the entries in $\mathbf{R}$. One primary purpose of finding such a latent representation is to produce out of sample predictions, i.e. predicting the ratings a user would give to movies they haven't seen given all the ratings made by everyone.

The simplest Matrix factorization generative model in [Salakhutdinov and Mnih(2008)] is

$$p(\mathbf{U}|\alpha_U) = \prod_{i=1}^{N} \mathcal{N}((U^T)_i | 0, \alpha_U^{-1}\mathbf{I}) \tag{1}$$

$$p(\mathbf{V}|\alpha_V) = \prod_{j=1}^{M} \mathcal{N}((V^T)_j | 0, \alpha_V^{-1}\mathbf{I}) \tag{2}$$

$$p(\mathbf{R}|\mathbf{U}, \mathbf{V}, \alpha) = \prod_{i=1}^{N} \prod_{j=1}^{M} \left[\mathcal{N}(R_{ij} | (U^T)_i V_j, \alpha^{-1})\right]^{I_{ij}} \tag{3}$$

where $X_i$ is the $i$th row of a matrix $\mathbf{X}$ and $I_{ij} = 1$ if a user rated movie $j$ and 0 otherwise. This model straightforwardly can be used to predict $R_{ij}$ for movies $j$ that user $i$ didn't originally rate.

A provided comma separated value (.csv) file contains ratings data for 7 movies and 10 users. The `movies.csv`[1] file contains three columns. The first is an integer user id. The

[1]Can be downloaded from `http://www.robots.ox.ac.uk/~fwood/anglican/teaching/mlss2014/bayesian_matrix_factorization/movies.csv`

second is an integer movie id. The third is the given integer rating. The data are a very small subset from `http://grouplens.org/datasets/movielens/`.

An example row is `9,1,2` which means user 9 rated movie 1 a 2. Note that user 9 did not rate movie 1 in the dataset.

The movies – (ID) – are

(1) Toy Story (1995)
(2) Twelve Monkeys (1995)
(3) Seven (Se7en) (1995)
(4) From Dusk Till Dawn (1996)
(5) Four Weddings and a Funeral (1994)
(6) Jurassic Park (1993)
(7) Pretty Woman (1990)

**Questions :**

**(1)** Complete the generative model code below corresponding to the generative model above starting from the following scaffolding and use it to predict the rating user 9 might give to movie 1.

```
[assume dot-product (lambda (u v)
                      (if (= (count u) 0) 0
                          (+ (* (first u) (first v))
                             (dot-product (rest u) (rest v)))))]
[assume repeatedly (lambda (N func)
    (if (= N 0)
        (list)
        (cons (func) (repeatedly (- N 1) func))))]

[assume D 4] ; this is the shared length of rows of U / columns of V
[assume sigma 0.01]
[assume sigma-U 1]
[assume sigma-V 1]
[assume get-row-U
    (mem (lambda (m) (repeatedly D <···>)))]
[assume get-col-V
    (mem (lambda (n) (repeatedly D <···>)))]

[observe-csv "movies.csv"
     (<···> (dot-product (get-row-U $1) (get-col-V $2)) <···>) $3]
[predict <···> ]
```

where <· · · > needs to be filled in by you.

**Answer**

```
[assume dot-product (lambda (u v)
                      (if (= (count u) 0) 0
                          (+ (* (first u) (first v))
                             (dot-product (rest u) (rest v)))))]
[assume repeatedly (lambda (N func)
    (if (= N 0)
        (list)
        (cons (func) (repeatedly (- N 1) func))))]

[assume D 4] ; this is the shared length of rows of U / columns of V
[assume sigma 0.01]
[assume sigma-U 1]
[assume sigma-V 1]
[assume get-row-U
    (mem (lambda (m) (repeatedly D (lambda () (normal 0 sigma-U) ))))]
[assume get-col-V
    (mem (lambda (n) (repeatedly D (lambda () (normal 0 sigma-V) ))))]

[observe-csv "movies.csv"
     (normal (dot-product (get-row-U $1) (get-col-V $2)) sigma) $3]
[predict  (dot-product (get-row-U 9) (get-col-V 1)) ]
```

**(2)** Change the generative model to be "open universe," i.e. make it allow for an unknown number of latent features.

**Answer** Change the line

```
[assume D 4]
```

to

```
[assume D (poisson 4)]
```

**(3)** Change the generative model to so that it does non-negative matrix factorization.

**Answer** An answer can be written by changing the observation model to be Poisson. The requires a positive rate argument which can be ensured by using Gamma priors for all entries of latent matrixes **U** and **V**.

```
[assume dot-product (lambda (u v)
                      (if (= (count u) 0) 0
                          (+ (* (first u) (first v))
                             (dot-product (rest u) (rest v)))))]
[assume repeatedly (lambda (N func)
    (if (= N 0)
        (list)
        (cons (func) (repeatedly (- N 1) func))))]

[assume D 4] ; this is the shared length of rows of U / columns of V
[assume sigma 0.01]
[assume sigma-U 1]
[assume sigma-V 1]
[assume get-row-U (mem (lambda (m)
                    (repeatedly D (lambda () (gamma 1 sigma-U) ))))]
[assume get-col-V (mem (lambda (n)
                     (repeatedly D (lambda () (gamma 1 sigma-V) ))))]

[observe-csv "movies.csv"
          (poisson (dot-product (get-row-U $1) (get-col-V $2))) $3]
[predict (poisson (dot-product (get-row-U 9) (get-col-V 1)))]
```

**(4)** Why might the generic inference strategies (RDB and PMCMC) not mix well for this model? Suggestion: diagram an execution trace tree.

**Answer** While randomness is generated lazily in this program, it remains the case that the first time a row of $\mathbf{U}^T$ and a column of $\mathbf{V}$ are needed to compute the mean for $R_{ij}$ *all* entries of those rows and columns are sampled simultaneously. In a sequential Monte Carlo inference algorithm the next time a weight is applied is when row $i$ or column $j$ is accessed again. The chance of sampling a good set of latent features from the prior is very low (i.e. one that yields high weights in all subsequent `observe`'s). For a single-site MCMC algorithm to really work a block update of a row or column needs to be performed. Modifying a single variable at once will, like usual, take a long time to converge.

## References

[Salakhutdinov and Mnih(2008)] Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pages 880–887. ACM, 2008.