Project 1: Hierarchical Poisson Factorization
An overview of building recommender systems with markovian inference

Mihail Feraru

December 2022

This project is based on the work presented in article "Scalable Recommendation with Poisson Factorization" [1] and it is implemented using *Gen.jl* framework, the code being available in the associated Jupyter Notebook.

1 Dataset selection and exploration

The source used to search for a suitable datasets was Kaggle. I found five appealing sets which were evaluated with respect to the following criteria: the size is manageable (less than 100.000 entries) and at the same time big enough (more than 10.000 entries), the entries should contain information about the time when they were generated, there should be multiple product categories and, finally, all ratings should be numerical. After this screening process I decided to continue with the **ModClocth ratings** [1] dataset. The detailed comparison can be viewed below in Table 1.

Dataset	Is size manageable?	Is big enough?	Contains timestamps?	Has multiple categories?	Are ratings numerical?
Amazon beauty	✗	✓	✓	✗	✓
Indonesia tourism	✓	✓	✗	✓	✓
Electronics eCommerce	✗	✓	✓	✓	✗
ModCloth ratings	✓	✓	✓	✓	✓
Amazon books	✗	✓	✗	✗	✗

Table 1: Dataset selection comparison

A lot of the columns present in the data are not relevant to our prediction model, so I decided to keep only a small subset of only five columns. A summary description of the columns can be found in Table 2. We see that there are around 1.000 items from 4 categories, 45.000 users and a heavy bias towards high ratings as the median is the same as the maximum value.

Column	Example	Type	Unique count	Minimum	Median	Maximum
item_id	7443	String	1020			
user_id	Alex	String	44784			
category	Dresses	String	4			
time	2010-01-21T08:00:00	DateTime	14741	2010-01-21T08:00:00	2016-01-26T08:00:00	2019-06-29T13:55:16.542
rating_score	4	Int64		1	5	5

Table 2: Dataset summary

A detailed distribution analysis confirms that more than 50% of user ratings have a score of 5 out of 5 and the same percentages can be observed across all clothing categories, showing the *positive bias of ratings* among people. The described statistics can be observed in Appendix A at Figure 5.

Additionally, based on the distribution of reviews grouped by user and item counts, which are presented in Appedinx A at Figure 6, I created some funny user personas and item types to have a better understanding of shopping habits of my users. [2]

[1] The dataset can be downloaded from Kaggle using the following URL: https://www.kaggle.com/datasets/ruchi798/marketing-bias-in-product-recommendations?select=modcloth.csv.

[2] Initially, I was thinking that we are going to do a live presentation of the project, so I was prepared to really "sell it" in a funny way.

1. **the causal buyer** has 5 or less reviews in the entire dataset but accounts for more than 60% of the total ratings; the hardest target for recommendations,
2. **the loyal client** generated between 5 and 55 reviews which represent 25% of the total; this type of client is the perfect one to make recommendations to,
3. **the shopping maniac** interacted with more than 55 items (up to the thousands) and accounts for about 15% of the total; they will buy almost anything.

The items can also be categorised in the following way:

1. **"my grandma' would wear" clothes** are the ones that almost no one reviewed, they have less than 50 interactions,
2. **trendy** ones obtained between 50 and 540 interactions,
3. **"must have"** items are bougth by almost everyone, they have more than 540 reviews.

An interesting fact is that *trendy* and *"must have"* items account for over 85% of the total number of reviews.

In further paragraphs, we will discuss the implementation and evaluation of HPF model, but we will ignore the **casual buyers** because they are the cause of the cold start problem. Also, we will ignore **"my grandma' would wear" clothes** because they are too unpopular to be recommended in a viabile way. For the other types of users we will aim to obtain a good value of **precision of top-k** and **recall of top-k** as those are suitable metrics for recommender systems.

2 HPF implementation and training procedure

Before implementing the model, I created a random data sampler which was used to select a small sanity check group of users and items with enough but not too many interactions between them. The sample consists of **5 users** with an activity of **at least 20 interactions** and **5 items** with a popularity of **at least 180**. The interactions were split in a train set consiting of the ones before the first day of **year 2017** and a validation set containing the rest after that date. There are **3 interactions** that should be predicted. The model must be able to beat a random recommender on this sanity check set on any selected metric. The described sanity check is presented as an interaction matrix in Table 3 below.

Users/Items	11960	16411	21296	22563	7443
Emma	5	**5**	5	2	2
Erin	**5**	4	5	**3**	5
Jo	4	5	0	0	0
Kayla	4	0	3	0	2
Tiffany	5	5	4	4	3

Table 3: Sanity check interaction matrix, each number being the rating a user gave to a item, 0 meaning that no interaction occurred, and the red ratings being the ones that should be predicted.

Mathematical and implementation details that are related to the standard HPF model were left out of this documentation, because I assume you being already familiar with them and there is no need of boring the reader. The implementation is a straightforward one in accordance with the mathematical model described in the paper and using *Gen's* generative functions.

The model was implemented in two different flavors: a naive one which performs relatively slow when trained and an optimized one which outperforms the former significantly. The following optimizations were implemented to improve the naive model:

1. all matrices were pre-allocated in order to reduce garbage collection and allocation overhead,
2. the generative function was split into multiple smaller ones for each vector of random variables,

3. the generative function was restricted to Gen's Static DSL.

In Figure 1 we see a comparison of running times with respect to the number of iterations, the size of K hyperparameter, and the size of the user-items sample. We see that with respect to the number of iterations there is a performance improvement with a decent constant factor, and while the naive model's running time grows quadratically when increasing K, the optimized model grows linearly. Unfortunately, both models have quadratic running times with respect to the matrix's size.

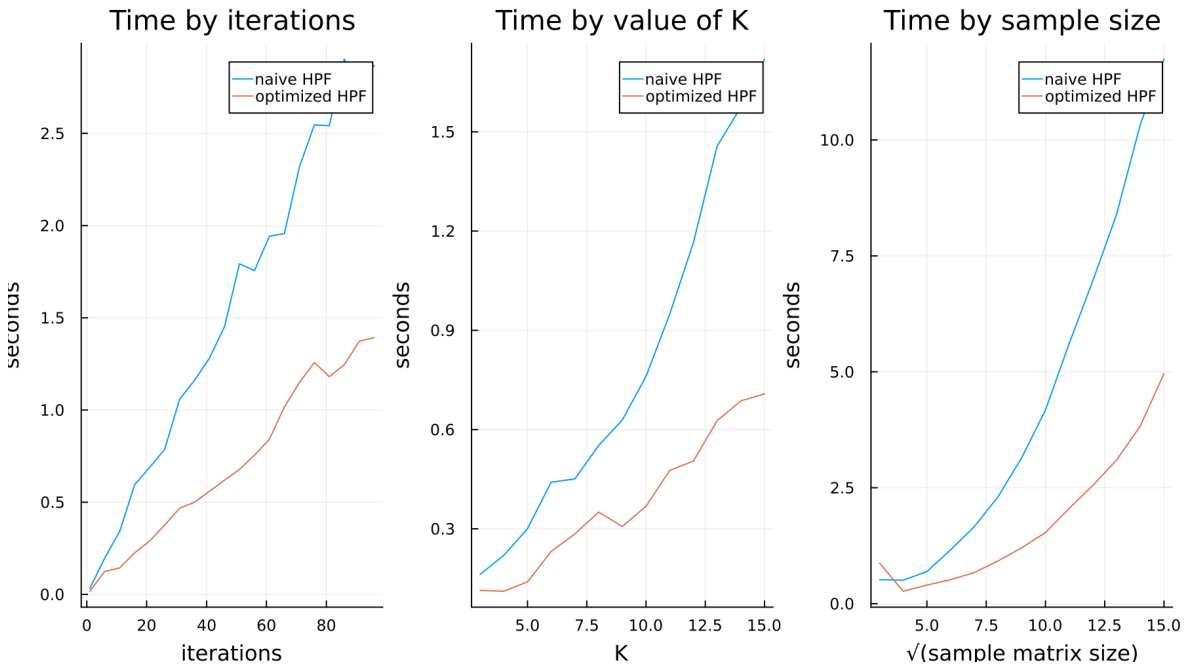

Figure 1: Comparison of running times of naive vs optimized HPF

The training procedure uses the **Metropolis-Hastings** algorithm to gradually learn what the best values for the latent variables are. Each latent variable is sampled individually and with respect to the dependence relation presented in the paper, so ξ_u and η_i get sampled first, θ_{uk} and β_{ik} afterwards. Additionally, a couple of traces are collected periodically during training in order to asses the convergence to the train set and the error of the predictions. In Figure 2 is displayed the *mean squared error (MSE)* during a training session on the sanity check sample using $K = 10$ and the default values for the other hyperparameters (presented in the paper). We see that the MSE converges after about 100-150 training epochs.

The prediction procedure takes the trained trace of the model and resamples a couple of θ and β matrices using Metropolis-Hastings and thinning (to reduce the correlation between samples), and then computes a mean over Poisson$(\theta_{uk}^T \cdot \beta_{ik})$ for each predicted value y_{ui}. Table 4 shows the evaludation of HPF on the sanity check set. It can be clearly seen that HPF outperforms random predictions of the validation set and also the decision of making no prediction at all.

3 Hyperparameter tuning and evaluation

To evaluate the performance of the model we will use three metrics: mean squared error, precision-k and recall-k. Precision-k and recall-k are the well known metrics but computed only for the first k items that were recommended to the user. For a proper evaluation of hyperparameters, I selected 3 random samples of 25 users and 25 items on which hyperparameter tuning was performed using simple searching. The tuned model was then trained and evaluated on a test set containing 50 users and 50 items randomly sampled.

In Table 5, is shown the search for a good value for K. It is clear that a greater K provides better performance and probably a much greater value (like 50 or 100) would be the peak performer, but as

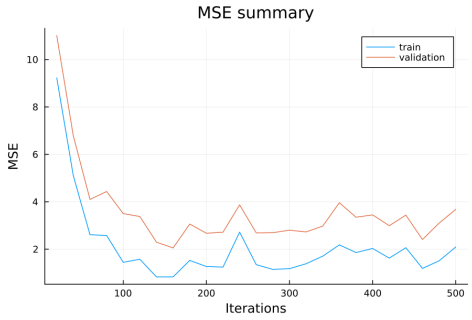

Figure 2: Train and validation MSE during training on the sanity check sample

Model	MSE
No prediction (validation)	2.36
Random (predict train and validation)	9.76
HPF (predict train and validation)	**2.72227**
Random (preload train, predict validation)	2.0
HPF (preload train, predict validation)	**1.51194**

Table 4: Comparison of random predictions and HPF, trained on the sanity check set, HPF clearly performing better

it is a proof-of-concept implementation and time constraints are important we will choose $K = 10$ as it offers a good trade-off between precision and recall in a reasonable time.

K	MSE	Precision k=5	Recall k=5	Time
1	6.646	13.866%	71.976%	9.6s
5	4.910	13.333%	72.531%	32.5s
10	**4.145**	**14.133%**	**76.976%**	**68.9s**
15	4.469	15.466%	74.007%	126.3s

Table 5: Gridsearch for hyperparameter K of HPF over 3 folds and 100 train epochs

After choosing a good value for K we perform a grid search for other hyperparameters, but not an exhaustive one, again because of time and computation constraints. In the Table 6, below, is shown a grid search for the following pairs of hyperparameters: $a' = c'$, $b' = d'$, and $a = c$. Even if the first pair of parameters provide the highest precision (over 27%), it causes a quite high degradation of the recall (around 10% lower than other parameters). Given so, I consider the set $a' = c' = 0.1$, $b' = d' = 0.5$, $a = c = 0.5$ to provide the best trade-off between precision and recall.

$a' = c'$, $b' = d'$, $a = c$	MSE	Precision k=5	Recall k=5
0.1, 1.0, 0.1	7.485	27.733%	61.793%
0.1, 0.5, 0.1	5.674	20.800%	72.515%
0.5, 1.0, 0.5	2.905	14.933%	75.087%
0.5, 0.5, 0.5	5.132	15.200%	76.420%
0.1, 0.5, 0.5	**5.538**	**25.333%**	**70.531%**
0.5, 0.5, 0.1	3.175	11.200%	67.000%

Table 6: Gridsearch for Gamma distributions hyperparameters over 3 folds and 100 iterations

Finally, the HPF model was trained on the test set for **600 iterations**, with **50 samples used for score prediction**, and a **thinnig value of 2** for about **1-1.5 hours**. A detailed plot of the MSE during training can be observed in Figure 3. After the training session, the model obtained the following results: a **MSE of 2.932**, **precision of top-5 of 14.800%**, and **recall of top-5 of 48.545**%.

4 Conclusions and further ideas

As a closing note, the results presented above prove that HPF is a suitable model for recommendation systems and it is able to achieve good accuracies. Looking at the evaluation on the test set, we can clearly see that it is considerably *harder* than the validation sets on which parameter tuning was performed. Probably the increase in dimensionality would require a greater K and a longer training time. On the other hand, the model predicted the top 5 prefered items with an accuracy of about

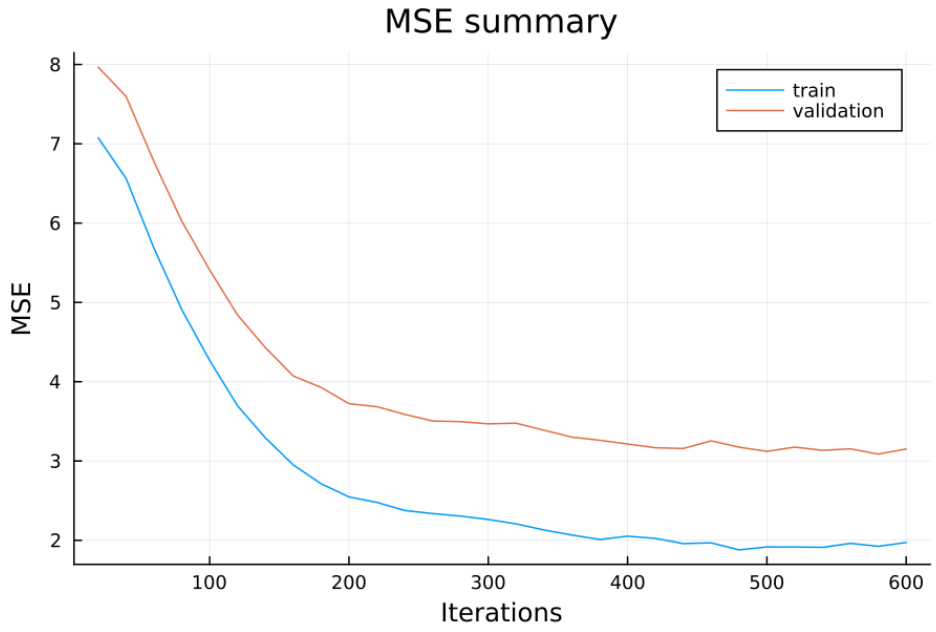

Figure 3: Train and validation MSE during training of the tuned model on the test set

14%, which considering the recall rate of almost 50% could result in a signifiant boost of sales in a real-world scenario as we anticipate what the user will buy and rate in 1 out of 2 cases.

Further improvements could include incorporating a proper variational inference implementation to speed-up and scale the training better, an exchaustive hyperparamter tunning, and an in-depth analysis of how the recommender can be tuned differently for each user persona or item type.

5 Extra: A failed approach of variational inference

I tried to implement *black-box variational inference* training using *black˙box˙vi* function from *Gen.jl* framework, but unfortunately it is not powerful enough, lacks sophisticated optimizers and it is not able to handle a hard problem like HPF. After 5000 iterations on a 2x2 matrix, the *evidence lower bound (ELBO)* reached a value of around -15, which is still too far away from a useful predicition and too slow comparted to Metropolis-Hastings. Figure 4 shows the evolution of ELBO during training and how slowly it progresses.

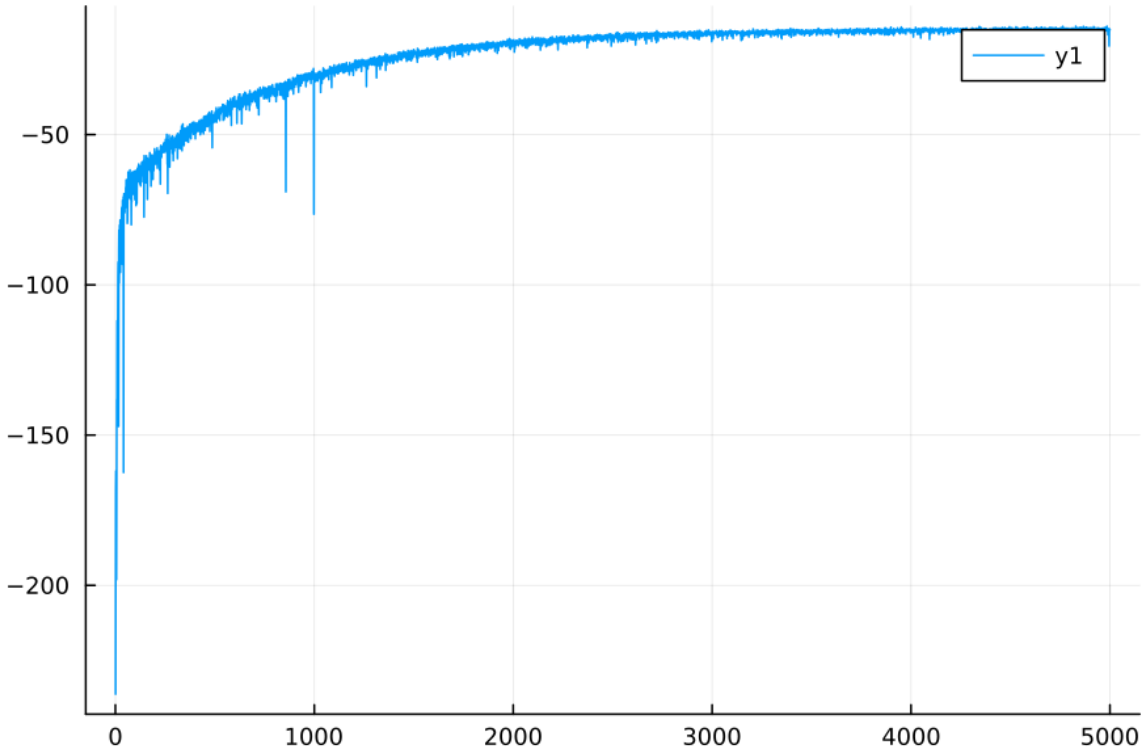

Figure 4: ELBO while training HPF using VI, converging very slowly

To illustrate how variational inference can be used with *Gen*, I implemented it for a simpler problem. Let's take the following arbitrary model:

$$B \sim \text{Beta}(\alpha, \beta), \quad G \sim \text{Gamma}(k, B * \theta)$$
$$y_i \sim \text{Beta}(xB, G^2)$$

We can define the variational parameters $\nu = \{\alpha', \beta', k', \theta'\}$ for the following variational family $q(B, G) = q(B|\alpha', \beta')q(G|k', \theta')$ which will be implemented in *Gen* as a generative function. In Figure 7 of Appendix A, can be observed a couple of points generated by the distribution above and the predicted distribution after using VI to learn the variational parameters and latent variables of the model. It is clear that the ELBO converges to a near-zero value very fast and the sampled distribution after VI resembels the distribution of the observed data.

References

[1] Prem Gopalan, Jake M. Hofman, and David M. Blei. *Scalable Recommendation with Poisson Factorization*. 2013. DOI: 10.48550/ARXIV.1311.1704. URL: https://arxiv.org/abs/1311.1704.

A Additional plots

On the following pages you can find the plots that were too big to fit alongside text. Maybe there are prettier ways of preseting them, but my LaTeX skills have limits.

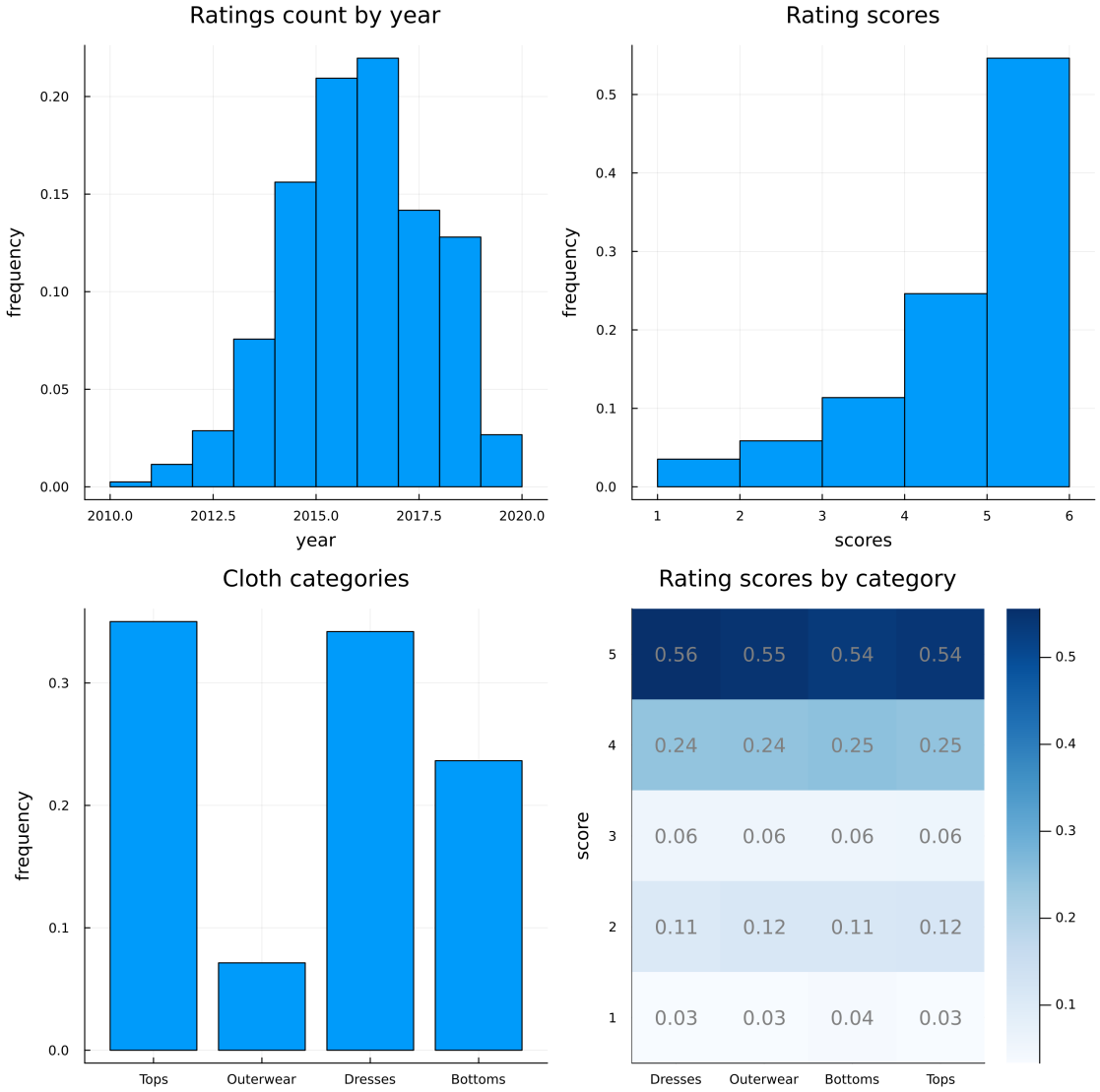

Figure 5: Distribution of ratings according to multiple criteria

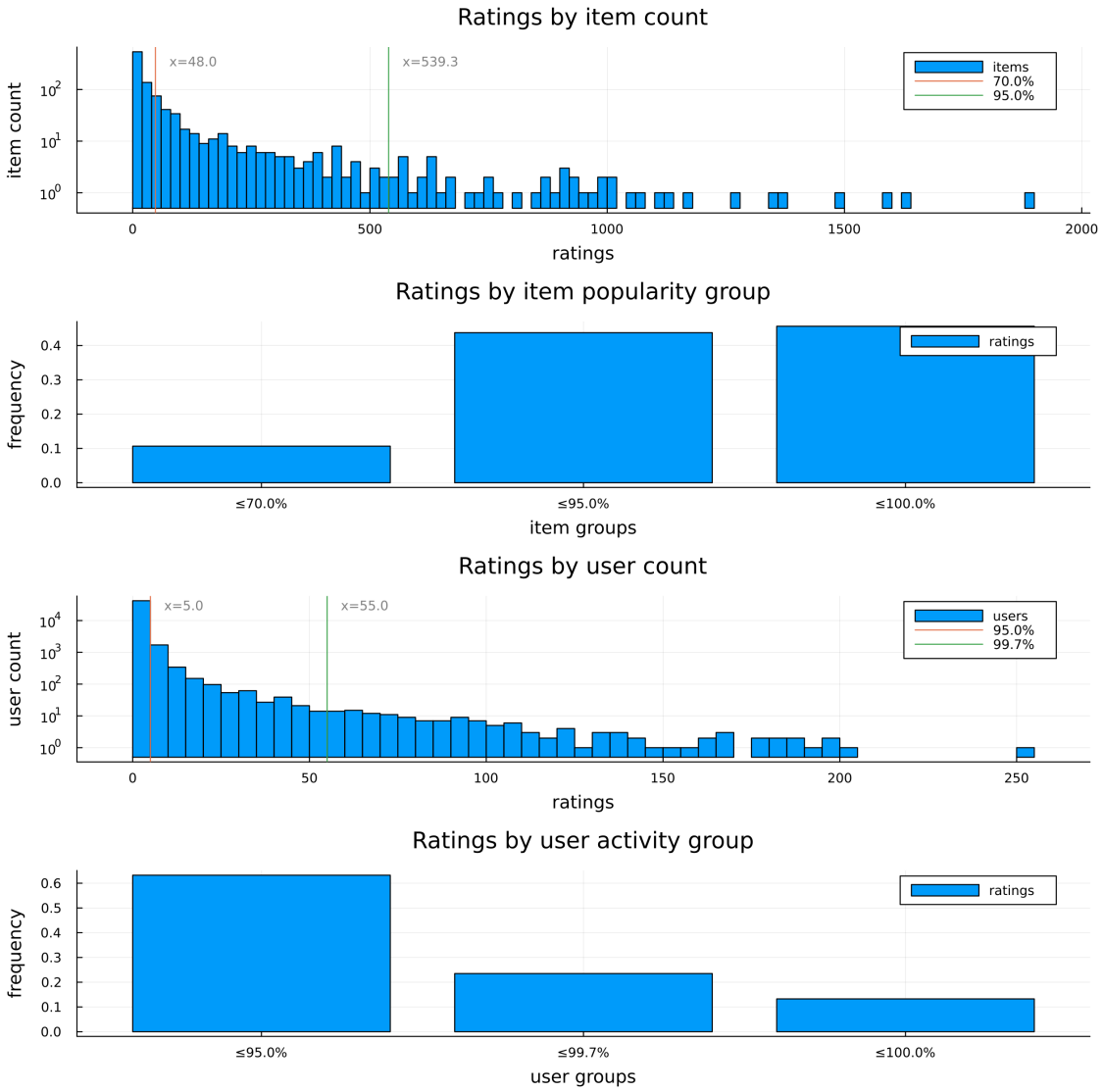

Figure 6: Distribution of user-item interactions by item popularity and user activity

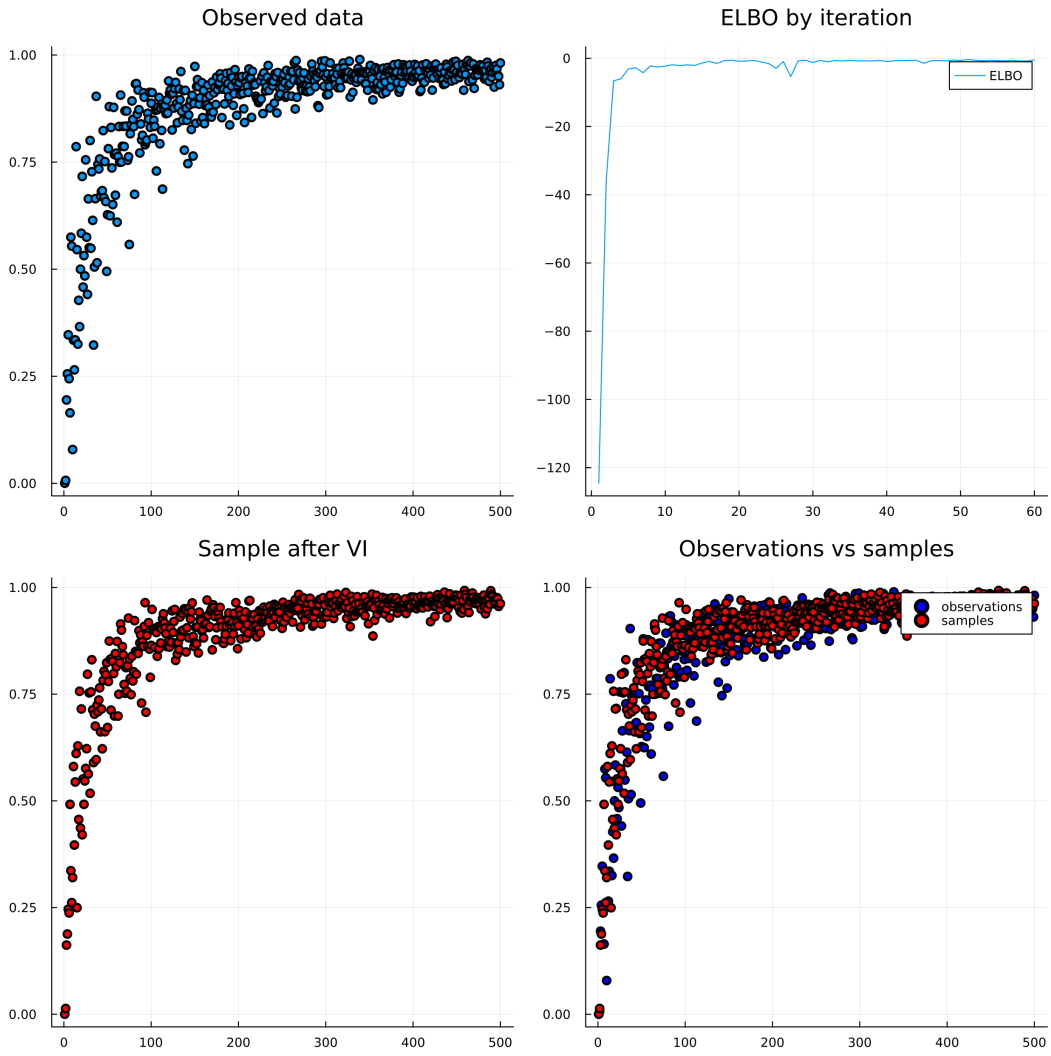

Figure 7: Example of variational inference on a custom model