Gibbs Sampler for Infinite Latent Feature Models

Bo Liu
Department of Statistical Science
Duke University
Durham, NC 27705
bl226@duke.edu

Linlin Li
Department of Statistical Science
Duke University
Durham, NC 27705
linlin.li434@duke.edu

Abstract

The Indian buffet process is a generative process that results in a probability distribution over equivalence classes of binary matrices with an unbounded number of columns. This distribution is often used as a prior in infinite latent feature models where the objects are represented by a potentially infinite array of features. In this paper, we introduce an implementation on a Gibbs sampler for inference in this model, perform optimizations and correctness check, and apply it to simulated and real datasets. Comparison with existing algorithms shows that a noticeable gain in performance is achieved.

Key words: Indian buffet process, binary matrix, infinite latent feature model, Gibbs sampler.

1 Background

Clustering is a set of methods that have been extensively studied and used in unsupervised learning since its first appearance in 1954 [5]. A large number of these methods assume that each observation belongs to one of some mutually exclusive clusters and attempt to classify the observations, although the classification can be deterministic (e.g. K means [19, 15]) or probabilistic (e.g. Latent Dirichlet Allocation [3]). However, many datasets do not intrinsically satisfy the assumption. Instead, they share multiple latent features and each observation contains a subset of these features. An example is in image detection where the aim is to identify and position items in images [23]. Multi-labeled data are also common in recommendation system, where each item is related to multiple tags [24].

All of these methods confront the critical challenge to determine how many latent features are necessary to model the hidden structure beyond the data. Usually, models with various number of latent features are tested and the optimal dimensionality is chosen based upon some metric on complexity or generalizaion error. In a Bayesian context, the dimensionality is controlled by both the prior belief and the data. Under this setting, the number of latent features can be infinite, and the data only reveal a finite subset of these features [18], such as Dirichlet process mixture models [2]. In a Dirichlet process mixture model, each object belongs to one of the latent classes, and each latent class defines a distribution over all latent features. The prior distribution of each class, which is a Dirichlet Process, is discrete with probability 1, but takes non-zero probability at infinite values [21], assuring Dirichlet process mixture models to have infinite underlying features [17]. Beta process can be the prior distribution on the probability of each latent feature if each observation has multiple features [10]. Similarly, such models also have infinite dimensionality in the latent features.

Dirichlet and Beta process have been studied as priors in non-parametric Bayesian modelling. Both processes can be reparameterized with stick breaking processes [4, 16]. Moreover, these processes can be specified by sequential processes, respectively, the Chinese restaurant process [11] and the Indian Buffet process [22].

Project Report for STA663 Statistical Computation, Duke University.

Ghahramani and Griffiths [8] describes how Indian buffet process can be used as a prior in statistical models where each object is represented by a subset of infinite features [also see 9]. In this paper, we implement a Gibbs sampler which generates Dirichlet process as a prior and samples from the posterior distribution.

We describe the algorithm in Section 2 and introduce the optimization on our implementation in Section 3. In Sections 4 and 5 we show how the algorithm can be applied to extract latent features in an image. The algorithm performs well with both simulated data and real data. We compare our implementation with other similar ones in Section 6, and conclude in Section 7.

2 Description of the algorithm

We implemented a Gibbs sampler on a linear-Gaussian binary latent feature model, where each observation is assumed to contain some of underlying features. The prior on the feature matrix is Gaussian, and the prior on the (binary) indicator matrix is a Dirichlet process. Our sampler is able to sample the indicators and the features from their posterior distributions.

2.1 Notations and conventions

Suppose $\mathbf{X}$ is an $N \times D$ matrix of objects, each row being an observation consisting of D components. The underlying latent features, which also lie in $\mathbb{R}^D$, are denoted as $\mathbf{A} \in \mathbb{R}^{K \times D}$, K being the total number of features revealed by the data. Note that K may vary during the sampling and there is a prior on it. $\mathbf{Z}$ is an $N \times K$ indicator matrix consisting of 0's and 1's, indicating whether an observation contains some certain feature.

Any vector without subscripts or superscripts should be considered a column vector unless specified. For any matrix $\mathbf{X}$, the i-th row is denoted as $\boldsymbol{x}_{i\cdot}$ and the j-th column is denoted as $\boldsymbol{x}_{\cdot j}$. The (i,j)-th entry of $\mathbf{X}$ is denoted as x_{ij}. $\boldsymbol{e}_k$ is a column vector with all entries being 0 except the k-th entry being 1. $\mathbf{E}_{ij}$ is a matrix with all entries being 0 except the (i,j)-th entry being 1. $\mathbf{I}$ is the identity matrix.

For Gibbs sampler, we use a number in bracket to indicate a sample in a specific iteration, (e.g. $\mathbf{Z}^{(t)}$ is the sampled matrix $\mathbf{Z}$ in the t-th iteration).

2.2 Priors and assumptions

- We define an Indian buffet process on the prior of $\mathbf{Z}$. The Indian buffet process scheme is introduced by Griffiths and Ghahramani [9, Sec 2.4]. Using $K_1^{(i)}$ to indicate the number of new dishes sampled by the i-th customer, the probability of any particular matrix being produce by the IBP is
$$P(\mathbf{Z} \mid \alpha) = \frac{\alpha^K}{\prod_{i=1}^N K_1^{(i)}!} \exp\{-\alpha H_N\} \prod_{k=1}^K \frac{(N-m_k)!(m_k-1)!}{N!}, \tag{1}$$
where $m_k = \sum_{i=1}^N z_{ik}$ is the number of objects possessing feature k, and
$$H_N = \sum_{n=1}^N \frac{1}{n}. \tag{2}$$
- The feature matrix has a Gaussian prior
$$\mathbf{A} \mid K, \sigma_A \sim \mathcal{MN}(\mathbf{O}_{K\times D}, \mathbf{I}_K, \sigma_A^2 \mathbf{I}_D). \tag{3}$$
- Given $\mathbf{Z}$ and $\mathbf{A}$, the distribution of $\mathbf{X}$ is Gaussian
$$\mathbf{X} \mid \mathbf{Z}, \mathbf{A}, \sigma_X \sim \mathcal{MN}(\mathbf{ZA}, \mathbf{I}_N, \sigma_X^2 \mathbf{I}_D). \tag{4}$$
- The prior distribution of α, σ_A and σ_X are gamma distributions.
$$\begin{aligned} \alpha &\sim \mathcal{G}(\alpha^a, \alpha^b); \\ \sigma_X &\sim \mathcal{G}^{-1}(\sigma_X^a, \sigma_X^b); \\ \sigma_A &\sim \mathcal{G}^{-1}(\sigma_A^a, \sigma_A^b). \end{aligned} \tag{5}$$
- If a column of $\mathbf{Z}$ are all 0's, this column as well as the corresponding row in $\mathbf{A}$ are removed, as no entries in this column will be assigned 1 in the Gibbs sampler.

2.3 Full conditionals

Griffiths and Ghahramani [9] noticed that $\mathbf{A}$ can be integrated out, so there is no need to update $\mathbf{A}$ each step.

$$p(\mathbf{X} \mid \mathbf{Z}, \sigma_X, \sigma_A) = \frac{1}{(2\pi)^{ND/2}\sigma_X^{(N-K)D}\sigma_A^{KD}|\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \frac{\sigma_X^2}{\sigma_A^2}\mathbf{I}|^{D/2}} \exp\left\{-\frac{1}{2\sigma_X^2}\mathrm{tr}(\mathbf{X}^{\mathrm{T}}(\mathbf{I} - \mathbf{Z}(\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \frac{\sigma_X^2}{\sigma_A^2}\mathbf{I})^{-1}\mathbf{Z}^{\mathrm{T}})\mathbf{X})\right\}. \quad (6)$$

- Full conditional for $\mathbf{Z}$:
  As the dimension of $\mathbf{Z}$ may change through the sampling, we need to find the full conditional for both the original and the (potential) additional columns of $\mathbf{Z}$.
  - For $k = 1, \cdots, K$,
    $$P(z_{ik} \mid \mathbf{X}, \mathbf{Z}_{-(i,k)}, \sigma_X, \sigma_A) \propto p(\mathbf{X} \mid \mathbf{Z}, \sigma_X, \sigma_A)P(z_{ik} \mid \boldsymbol{z}_{-i,k}), \quad (7)$$
    where $P(z_{ik} = 1 \mid \boldsymbol{z}_{-i,k}) = \frac{m_{-i,k}}{N}$.
    $m_{-i,k}$ denotes the number of objects possessing feature k, excluding i.
  - There might be K_i^{new} columns added to $\mathbf{Z}$, each additional column being $\boldsymbol{e}_i$.
    $$P(K_i^{\text{new}} \mid \mathbf{X}, \mathbf{Z}, \sigma_X, \sigma_A, \alpha) \propto p(\mathbf{X} \mid \mathbf{Z}_{i,K_i^{\text{new}}}^{+}, \sigma_X, \sigma_A)P(K_i^{\text{new}} \mid \alpha), \quad (8)$$
    where $K_i^{\text{new}} \mid \alpha \sim \text{Possion}(\alpha/N)$,
    $$\mathbf{Z}_{i,K_i^{\text{new}}}^{+} = (\mathbf{Z}, \underbrace{\boldsymbol{e}_i, \cdots, \boldsymbol{e}_i}_{K_i^{\text{new}}\text{ columns}}). \quad (9)$$
- Full conditional for α:
  $$\alpha \mid \mathbf{Z} \sim \mathcal{G}(\alpha^a + K, \alpha^b + H_N). \quad (10)$$
- Full conditional for σ_X and σ_A:
  $$\begin{aligned} p(\sigma_X \mid \mathbf{X}, \mathbf{Z}, \sigma_A) &\propto p(\mathbf{X} \mid \mathbf{Z}, \sigma_X, \sigma_A)p(\sigma_X), \\ p(\sigma_A \mid \mathbf{X}, \mathbf{Z}, \sigma_X) &\propto p(\mathbf{A} \mid \mathbf{Z}, \sigma_X, \sigma_A)p(\sigma_X). \end{aligned} \quad (11)$$

2.4 Gibbs sampler

Algorithm 1: Gibbs sampler for linear-Gaussian binary latent feature model

Data: An $N \times D$ matrix $\mathbf{X}$

Input: Number of iterations T, initial values of $\alpha^{(0)}$, $\sigma_X^{(0)}$, $\sigma_A^{(0)}$, and hypermeters α^a, α^b, σ_X^a, σ_X^b, σ_A^a, σ_A^b

Output: Results of $\mathbf{Z}$, α, σ_X, σ_A in each iteration

Sample $\mathbf{Z}^{(0)}$ from Indian buffet process (Appendix A);
for $t = 1$ *to* T **do**
  for i *in randomized* $1 : N$ **do**
    for $k = 1$ *to* K **do**
      Sample $z_{ik}^{(t)} \mid \mathbf{X}, \mathbf{Z}_{-(i,k)}^{(t-1)}, \sigma_X^{(t-1)}, \sigma_A^{(t-1)}$;
    end
    Sample $K_i^{\text{new},(t)} \mid \mathbf{X}, \mathbf{Z}^{(t)}, \sigma_X^{(t-1)}, \sigma_A^{(t-1)}, \alpha^{(t-1)}$;
  end
  Sample $\alpha^{(t)} \mid \mathbf{Z}^{(t)}$;
  Sample $\sigma_X^{(t)} \mid \mathbf{X}, \mathbf{Z}^{(t)}, \sigma_A^{(t-1)}$;
  Sample $\sigma_A^{(t)} \mid \mathbf{X}, \mathbf{Z}^{(t)}, \sigma_X^{(t)}$;
end

3 Optimization

In code optimization and speeding, an important philosophy is given by the Amdahl's law [1] and the Gustafson's law [12]. Both laws state that the upper bound of theorical speedup on a part of a system depends on the frequency this part is used. In parallel programming, these laws also show how much acceleration can be expected after dispatching the workload onto multiple cores.

Guided by these laws, we prioritized the optimization of code based on the frequency of execution and the maximum possible speedup ratio. We first design algorithms that have lower asymptotic time complexity. Afterwards, we tried optimization based on numba, cython, which is implemented in static programming languages such as C or C++.

The `numpy` functions highly rely on various linear algebra libraries such as LAPACK. The complexity of basic matrix operations are prescribed by BLAS. From these standards, the time complexity of some matrix operations are listed in Table 1.

We may assume that $K \ll N$ (Assumption 1) as we would not expect more features to be extracted than the number of observations. Unless under high-dimensional situations, we might further assume that we always have a relatively large amount of data such that $N > D$ (Assumption 2).

Table 1: Time complexity of some `numpy` matrix operations

Operation	Input Shape	Output Shape	Estimated Time Complexity
Matrix Multiplication	$(m,n),(n,k)$	(m,k)	$O(mnk)$
Matrix Inverse	(n,n)	(n,n)	$O(n^3)$
Determinant	(n,n)	(1)	$O(n^3)$
Singular Value Decomposition	(m,n)	$(m,n),(n,n),(n,n)$	$O(mn\min\{m,n\})$
Trace	(n,n)	(1)	$O(n)$
L_2 Norm	(n)	(1)	$O(n)$
Frobenius Norm	(m,n)	(1)	$O(mn)$

3.1 Optimization on calculating the conditional likelihood

The function `lp` is intended to calculate the conditional likelihood of $\mathbf{X}$ given $\mathbf{Z}, \sigma_X, \sigma_A$. This calculation is called in each draw of z_{ik}, σ_X and σ_A. Moreover, it is called multiple times in determining K_i^{new}. In each iteration, `lp` is called $N(K+c)+2$ times, c being a constant which does not vary with N, K or D. Here and henceforth, we will use the big-O notation and the number of `lp` calls is $O(NK)$.

For numerical stability, we calculate the log conditional likelihood instead, which is given by

$$\log p(\mathbf{X} \mid \mathbf{Z}, \sigma_X, \sigma_A) = -\frac{ND}{2}\log(2\pi) - (N-K)D\log\sigma_X - KD\log\sigma_A - \frac{D}{2}\log\left|\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \frac{\sigma_X^2}{\sigma_A^2}\mathbf{I}\right| - \frac{1}{2\sigma_X^2}\mathrm{tr}\left\{\mathbf{X}^{\mathrm{T}}\left(\mathbf{I} - \mathbf{Z}\left(\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \frac{\sigma_X^2}{\sigma_A^2}\mathbf{I}\right)^{-1}\mathbf{Z}^{\mathrm{T}}\right)\mathbf{X}\right\}. \quad (12)$$

Suppose the singular value decomposition of $\mathbf{Z}$ is

$$\mathbf{Z} = \mathbf{U}\mathrm{diag}(\boldsymbol{d})\mathbf{V}^{\mathrm{T}}, \quad (13)$$

where $\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}$, $\mathbf{V}^{\mathrm{T}}\mathbf{V} = \mathbf{V}\mathbf{V}^{\mathrm{T}} = \mathbf{I}$. Then,

$$\left|\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \frac{\sigma_X^2}{\sigma_A^2}\mathbf{I}\right| = \prod_{i=1}^{\min\{N,K\}}\left(d_i^2 + \frac{\sigma_X^2}{\sigma_A^2}\right), \quad (14)$$

$$\mathrm{tr}\left\{\mathbf{X}^{\mathrm{T}}\left(\mathbf{I} - \mathbf{Z}\left(\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \frac{\sigma_X^2}{\sigma_A^2}\mathbf{I}\right)^{-1}\mathbf{Z}^{\mathrm{T}}\right)\mathbf{X}\right\} = \mathrm{tr}(\mathbf{X}^{\mathrm{T}}\mathbf{X}) - \sum_{i=1}^{\min\{N,K\}}\frac{d_i^2}{d_i^2 + \sigma_X^2/\sigma_A^2}||\boldsymbol{u}_{\cdot i}^{\mathrm{T}}\mathbf{X}||_2^2. \quad (15)$$

The proof is provided in Appendix B.1. The value of $\mathrm{tr}(\mathbf{X}^{\mathrm{T}}\mathbf{X})$ can be calculated and stored before the iteration steps, and thus the time on calculating this term is negligible. The comparison on time complexity of calculating the conditional likelihood is shown in Table 2.

Table 2: Time complexity comparison of `lp`

	Time Complexity	Under Assumption 1	Under Assumption 2
Original	$O(N^2D + ND^2 + N^2K + NK^2 + K^3)$	$O(N^2D + ND^2)$	$O(N^2D)$
Improved	$O(N(K+D)\min\{N, K\})$	$O(NDK)$	$O(NDK)$

3.2 Optimization on sampling K_i^{new}

In each iteration, this procedure is executed N times. This is smaller than `lp` by a multipler of K. However, in most applications where the number of latent features are expected to be low, this procedure takes comparative amount time with `lp` (excluding calls on `lp` within the procedure).

Although we can express the posterior as proportional to the multiplication of likelihood and prior, it is rather difficult to find the normalization term as it involves an infinite sum which does not have a closed form result. In practice, we use a truncated distribution instead; set a large upper bound C for the value of K_i^{new} and calculate the posterior probability of each value. The prior, which is a poisson distribution, is easy and fast to calculate. Hence, it is important to speed up finding the likelihood.

Definitely, we can use the improved version in Section 3.1. However, we may note that the matrices $\mathbf{Z}$ passed into `lp` are related, in the sense that for each increment of K_i^{new}, $\mathbf{Z}$ is appended by a column $\boldsymbol{e}_i$ on the right.

Let $\widetilde{\mathbf{Z}} = [\mathbf{Z} \quad \boldsymbol{e}_i]$, $\mathbf{W} = \mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \frac{\sigma_X^2}{\sigma_A^2}\mathbf{I}$, $\mathbf{\Gamma} = \mathbf{Z}(\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \frac{\sigma_X^2}{\sigma_A^2}\mathbf{I})\mathbf{Z}^{\mathrm{T}}$, $t = \mathrm{tr}\{\mathbf{X}^{\mathrm{T}}(\mathbf{I} - \mathbf{Z}(\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \frac{\sigma_X^2}{\sigma_A^2}\mathbf{I})^{-1}\mathbf{Z}^{\mathrm{T}})\mathbf{X}\}$, and define $\widetilde{\mathbf{W}}, \widetilde{\mathbf{\Gamma}}, \widetilde{t}$ similarly. For simplicity of notation, we define $\mu = 1 + \frac{\sigma_X^2}{\sigma_A^2} - \gamma_{ii}$. It can be shown (see Appendix B.2) that

$$|\widetilde{\mathbf{W}}| = \mu|\mathbf{W}|, \ \widetilde{\boldsymbol{\gamma}}_{\cdot i} = \boldsymbol{\gamma}_{\cdot i} + \mu^{-1}(\gamma_{ii} - 1)(\boldsymbol{\gamma}_{\cdot i} - \boldsymbol{e}_i), \tag{16}$$

and

$$\widetilde{t} = t - \mu^{-1}||\mathbf{X}^{\mathrm{T}}\boldsymbol{\gamma}_{\cdot i} - \boldsymbol{x}_{i\cdot}||_2^2. \tag{17}$$

The comparison on time complexity of sampling K_i^{new} is shown in Table 3.

Table 3: Time complexity comparison of sampling K_i^{new}

	Time Complexity	Under Assumption 1	Under Assumption 2
Original	$O(CN^2D + CND^2 + CN^2K + CNK^2 + CK^3)$	$O(CN^2D + CND^2)$	$O(CN^2D)$
Improved `lp`	$O(CN(K+D)\min\{N, K\})$	$O(CNDK)$	$O(CNDK)$
Recursive	$O(NK\min\{N, K\} + CND)$	$O(NK^2 + CND)$	$O(NK^2 + CND)$

3.3 Execution time comparison

We tested the accuracy of both optimizations, and recorded the respective execution time. The discrepancy between optimized version and the original version is at the scale of 10^{-15}, which is quite satisfactory. The results on running time are shown as in Table 4 and Table 5. The calculation of log conditional likelihood is required in updating each element of $\mathbf{Z}$ and the Metropolis steps for σ_X and σ_A. Therefore, the efficiency brought by optimizing `lp` is significant. In the process of sampling new features, our proposed recursive algorithm outperforms the original algorithm even with `lp` optimized. Compared to the vanilla implementation, this procedure leads to tremendous efficiency promotion.

3.4 Optimization through parallelism

Gibbs samplers generate dependent samples from a Markov Chain. As the n-th iteration is based upon the $(n-1)$-th iteration, it is unlikely feasible to parallel the sampling process into multiple cores. Although matrix operations generally can be performed in parallel, `numpy` module is already

Table 4: Real execution time comparison of Section 3.1

N	D	K	Old time (s)		New time (s)		Avg.
			Avg.	Max.	Avg.	Max.	Speedup
200	40	10	0.000307	0.001350	0.000222	0.002648	1.383
		20	0.000386	0.003899	0.000254	0.004414	1.520
	80	20	0.000378	0.001547	0.000227	0.001635	1.665
		40	0.000642	0.003046	0.000341	0.001650	1.883
1000	200	50	0.008888	0.024496	0.001331	0.009833	6.678
		100	0.018554	0.029955	0.002945	0.007445	6.300
	400	100	0.022805	0.037314	0.003169	0.005084	7.196
		200	0.057706	0.069650	0.007470	0.011485	7.725
5000	1000	250	1.409483	1.515692	0.051928	0.074056	27.143
		500	5.540976	6.007763	0.138402	0.210703	40.035
	2000	500	6.044738	6.234605	0.158751	0.195732	38.077
		1000	29.506844	29.719376	0.472647	0.633464	62.429
10000	2000	500	13.144238	13.335559	0.313577	0.378531	41.917
		1000	61.384636	61.649799	0.904890	1.051028	67.837
	4000	1000	64.492656	64.732630	1.325637	1.542795	48.650
		2000	—— *	—— *	4.822996	5.329228	——

All combinations of hyperparameters are tested for 100 repetitions.
* Unable to calculate 100 repetitions within 24 hours.

Table 5: Real execution time comparison of Section 3.2

N	D	K	Old time (s)		New time (s)		Avg.
			Avg.	Max.	Avg.	Max.	Speedup
200	40	10	0.002723	0.010516	0.000570	0.001259	4.777
		20	0.003887	0.007241	0.000530	0.001176	7.334
	80	20	0.003093	0.005747	0.000542	0.001441	5.707
		40	0.004878	0.010489	0.000796	0.001394	6.128
1000	200	50	0.013081	0.023041	0.001723	0.003313	7.592
		100	0.035016	0.056258	0.005390	0.010518	6.496
	400	100	0.033598	0.054375	0.004077	0.005322	8.241
		200	0.078245	0.094424	0.010370	0.011971	7.545
5000	1000	250	0.426222	0.501770	0.047695	0.055292	8.936
		500	1.155086	1.319234	0.146501	0.164900	7.884
	2000	500	1.657428	1.967195	0.179803	0.228871	9.203
		1000	4.538577	6.106669	0.554646	0.653897	8.183
10000	2000	500	3.214276	3.844124	0.320323	0.426812	10.034
		1000	7.758868	8.501177	0.980188	1.212802	7.916
	4000	1000	10.173895	11.740906	1.022655	1.247130	9.949
		2000	37.213665	42.941216	4.901111	5.721428	7.593

All combinations of hyperparameters are tested for 10 repetitions, with $C = 5$.
The optimized version of `lp` is called in the original version of sampling K_i^{new} to exclude the speedup achieved from `lp`.

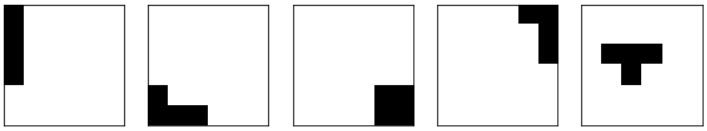
highly optimized by utilizing cache memory and assembler implementation. Moreover, many architectures now enable BLAS to take advantage of a multicore machine. Thus, we did not optimize the code further through parallelism.

3.5 Optimization via C++

In many cases, the interpretation time in Python can be reduced by ahead-of-time compilation (which can be done through modules, e.g. `numba`[13]), explicitly declaring variable types and turn off checks where safety is assured. However, the precompiled version was unexpectedly slow. The reason might be that most of `numpy` functions are implemented in C/C++ and there is little advantage of rewriting built-in functions in risk of numerical instability (Table 6).

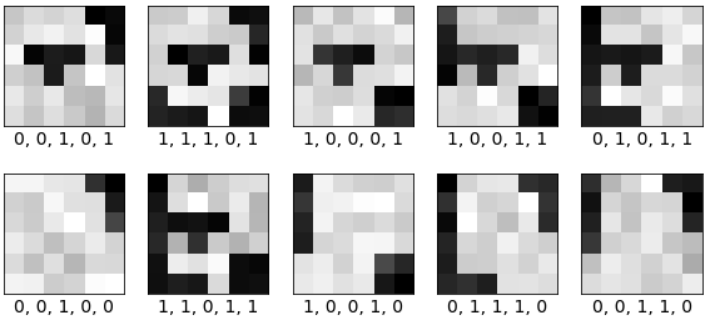
Table 6: The execution time of three implementations of `lp` with $N = 1000$, $D = 400$ and $K = 200$ with 1000 repetitions, shown in seconds and relative ratios.

	Original Version	Improved Version	Numba Version
Avg.	0.029838s (1.000)	0.015699s (0.526)	0.156077s (5.231)
Max.	0.126699s (1.000)	0.069471s (0.548)	0.428310s (3.381)



(a) Basic tetris-like patterns

(b) Features extracted from the data



(c) Features detected in images

Figure 1: (b) Results of extracted features from the Gibbs sampler. These features perfectly coincide with the basic patterns, ignoring background noise. (c) Ten images are randomly selected and the labels correspond to latent features detected from the Gibbs sampler.

4 Simulation

We performed a simulation study on a dataset of images generated from basic five tetris-like patterns (Fig 1a). Each image consists of one or multiple patterns and is disturbed by a random noise. Each pattern is embedded in a 6×6 matrix which can be reshaped into a 36-dimension vector. To elaborate it mathematically, let $\boldsymbol{f}_1, \cdots, \boldsymbol{f}_5$ represent the patterns in vector forms. The images $\{\boldsymbol{x}_i\}_{i=1}^N$ are independently generated by

$$\boldsymbol{x}_i = \sum_{j=1}^{5} c_j \boldsymbol{f}_j + \boldsymbol{e}_i, \tag{18}$$

where $c_j \in \{0, 1\}$ are not all zero and $\boldsymbol{e}_i \sim \mathcal{N}(\boldsymbol{0}, 0.01\mathbf{I})$.

The true value of $\sigma_X = 0.1$ under this setting. The parameters are initialized with $\sigma_X = \sigma_A = \alpha = 1$ and updated via Monte Carlo and Metropolis steps. The Gibbs sampler went through 1000 iterations, and stabilized in approximately 400 iterations (Fig 2). The algorithm found 5 latent features which perfectly indicated the presence and absence of the five patterns.

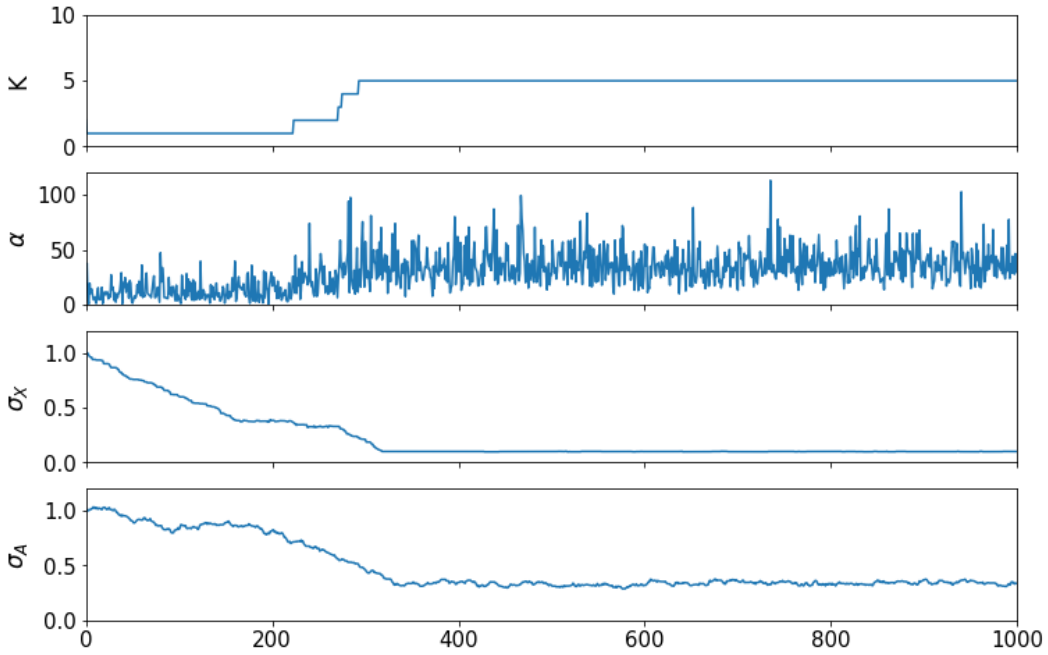
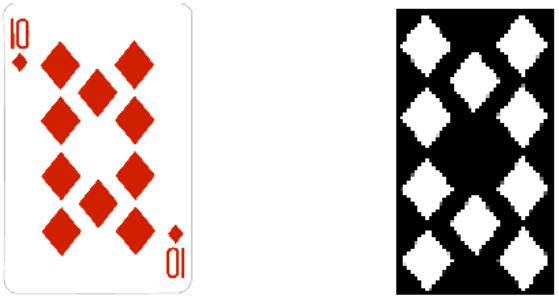


Figure 2: Trace plots for the dimensionality and parameters over 1000 iterations of sampling. The posterior estimation of $\sigma_X = 0.0996$ is quite close to the true value 0.1.

5 Latent features in poker card images

We applied this Gibbs sampler on a dataset consisting of 1056×691 pixel PNG images of poker cards. We selected out 100 images from the suit of diamonds excluding Jack, Queen and King, cropped and downsampled each image to 80×44 pixels in grayscale. We represented each image $\boldsymbol{x}_i$ by a flattened vector of 3520 dimensions. An example of an original image and the corresponding processed image is shown in Fig 3. The Gibbs sampler was initialized with $\sigma_X = 0.15$, $\sigma_A = 0.25$ and $\alpha = 1$, and stabilized after approximately 100 iterations. We extracted the mean image in observance of the zero-mean prior on the features, and 14 other latent features were suggested (Fig 4). The latent features, along with the reconstructed images are shown in Fig 5.



(a) Original image (b) Processed image

Figure 3: The original and processed image of ten of diamonds (10♦)

Although it is hard to interprete all these latent features, it is delighting to see that the features are able to capture the symmetry in most card images and reveal some particular patterns. For example, feature 1 (excluding the mean image) captures the pattern of a diamond in the center (light) with no diamonds in the corners (dark), which is shared by A♦ and 3♦. It is also interesting to note some enhancing-declining pairs in these features. Feature 7 and 8 have opposite effects, as a dark diamond in the center removes it from 4♦ and a light one enhances it on 5♦. Similarly goes for

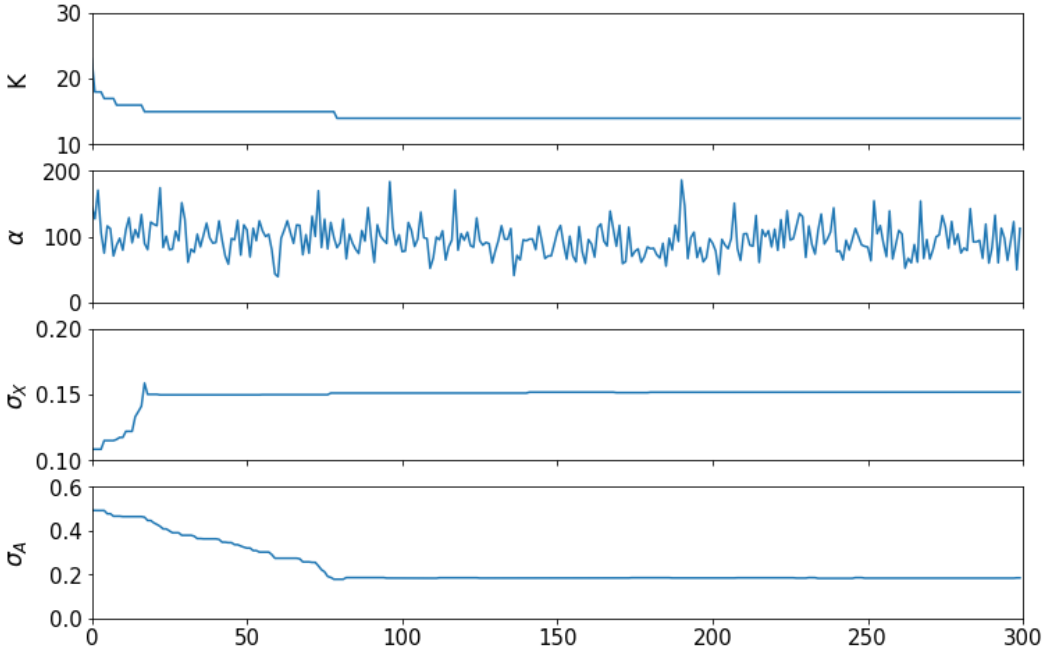


Figure 4: Trace plots for the dimensionality and parameters over 300 iterations of sampling.

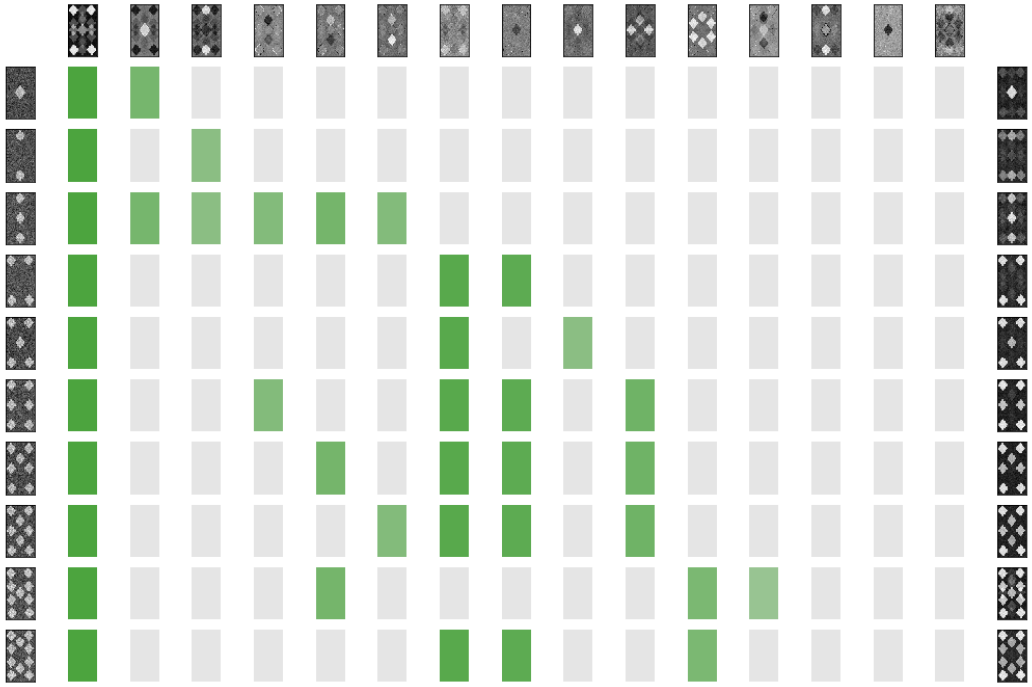


Figure 5: Feature matrix in the left-ordered form. The mean image (left) and 14 features are shown on the top. In the left-most column are the images for the first suit of cards, and the images reconstructed from the features are displayed on the right respectively. The rectangles in the middle indicate which features are possessed by the images, with colored rectangles showing possession and the gray ones showing non-possession. The shades of color indicate the frequency of that feature being possessed. Both indicators and the reconstructed images are based upon the 300th sample.

feature 3 through 5. These features demonstrate a pattern of two diamonds vertically arranged in the

middle. Feature 3 declines both diamonds while feature 5 enhances them. Feature 4, which has both a light and a dark diamond, enhances the one on the top and declines the other at the bottom. These features contribute to the difference among 6♦, 7♦ and 8♦.

6 Comparative Analysis

Feature extraction, which involves reducing the dimensionality of original data, plays a significant role in data analysis. Through feature extraction, data in high dimensionality are embedded into a low dimension space while still being described with sufficient accuracy. Many feature extraction techniques have been proposed, from traditional methods on low-rank matrix approximation (e.g. PCA) to neural network based algorithms (e.g. autoencoder). A majority of these methods, however, requires predetermined hyperparameters controlling the number of features. On the poker card dataset, principal component analysis (PCA) requires 49 features to maintain information akin to our method (measured in variance). Although as few as 8 components are sufficient to reconstruct the images with differences imperceptable to human eyes, determination on this number requires domain knowledge. Our methods extracted 14 features without manual selection, which is slightly more redundant than 8. However, it outperforms much over 49 features suggested by PCA comparably in absense of manual selection.

Some other implementations can be found on the same algorithm as introduced in this paper, one of which is given by Chai [6]. The authors of that implementation was able to reduce the running time by utilizing the one-rank update of matrix inversion [9], they failed to exploit sufficiently for potential speedup possibility. Furthermore, the code was not written in OOP (object oriented programming), nor was it well wrapped for reproduction.

The execution time of these implementations was recorded respectively under the same dataset. The profiling information is given in Table 7.

Table 7: Profile on simulated data introduced in Section 4

	Implementation of Chai [6]			Our implementation		
	Total time (sec)	# Exec	Avg. time (μsec)	Total time (sec)	# Exec	Avg. time (μsec)
Sample $\mathbf{Z}$	450.754	970,000	464.694	140.783	490,218	287.184
Sample K	82.614	97,000	851.690	21.080	97,000	217.319
Sample α	0.140	1,000	139.871	0.028	1,000	28.423
Sample σ_X	0.348	1,000	347.566	0.248	1,000	248.225
Sample σ_A	0.331	1,000	331.169	0.221	1,000	221.456
Others	2.386	———	———	1.821	———	———
Total	536.572	———	———	164.182	———	———

7 Discussion

We have demonstrated that our implementation of the algorithm proposed by Griffiths and Ghahramani [9] achieved satisfactory accuracy and efficiency. Also, we applied the algorithm on datasets where the observations are exchangeable and each observation possesses a finite subset of infinite latent features, and obtained reasonable and interpretable results. The strategy of taking the limit of a finite model from latent classes to infinite features provides a direction for a wild class of latent structures, such as in Teh [20] and Chen et al. [7].

Although the algorithm we implemented upon extended the models from definite latent classes to a set of latent features, limiting $\mathbf{Z}$ within the class of binary matrices may not be appropriate for many datasets. This constraint can be relaxed by allowing $\mathbf{Z}$ to take on natural numbers (count of features) or real numbers (proportion of features). Furthermore, the model is sensitive to location and scale. Identical patterns appearing in various position and size are not considered the same feature, which entangles its application in object detection or pattern recognition. Further work can be done on the improvement of the algorithm.

References

[1] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS 67 (Spring), pages 483–485, New York, NY, USA, 1967. Association for Computing Machinery. ISBN 9781450378956. doi: 10.1145/1465482.1465560. URL https://doi.org/10.1145/1465482.1465560.

[2] Charles E Antoniak. Mixtures of dirichlet processes with applications to bayesian nonparametric problems. *The annals of statistics*, pages 1152–1174, 1974.

[3] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[4] Tamara Broderick, Michael I Jordan, Jim Pitman, et al. Beta processes, stick-breaking and power laws. *Bayesian analysis*, 7(2):439–476, 2012.

[5] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 132–149, 2018.

[6] Christine P. Chai. Sta663-christine-chai-final-project: Implementation of the indian buffet process (ibp), 2015. URL https://github.com/star1327p/STA663-Christine-Chai-Final-Project.

[7] Bo Chen, Gungor Polatkan, Guillermo Sapiro, David B Dunson, and Lawrence Carin. The hierarchical beta process for convolutional factor analysis and deep learning. 2011.

[8] Zoubin Ghahramani and Thomas L Griffiths. Infinite latent feature models and the indian buffet process. In *Advances in neural information processing systems*, pages 475–482, 2006.

[9] Thomas L Griffiths and Zoubin Ghahramani. Infinite latent feature models and the indian buffet process. 2005.

[10] Thomas L. Griffiths and Zoubin Ghahramani. The indian buffet process: An introduction and review. *Journal of Machine Learning Research*, 12(32):1185–1224, 2011. URL http://jmlr.org/papers/v12/griffiths11a.html.

[11] Thomas L Griffiths, Michael I Jordan, Joshua B Tenenbaum, and David M Blei. Hierarchical topic models and the nested chinese restaurant process. In *Advances in neural information processing systems*, pages 17–24, 2004.

[12] John L. Gustafson. Reevaluating amdahls law. *Commun. ACM*, 31(5):532533, May 1988. ISSN 0001-0782. doi: 10.1145/42411.42415. URL https://doi.org/10.1145/42411.42415.

[13] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM 15, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450340052. doi: 10.1145/2833157.2833162. URL https://doi.org/10.1145/2833157.2833162.

[14] Bo Liu and Linlin Li. Infinite-latent-feature-models-and-the-indian-buffet-process, 2020. URL https://test.pypi.org/project/IBPILFM.

[15] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28 (2):129–137, 1982.

[16] John Paisley, David Blei, and Michael I Jordan. The stick-breaking construction of the beta process as a poisson process. *arXiv preprint arXiv:1109.0343*, 2011.

[17] Carl Edward Rasmussen. The infinite gaussian mixture model. In *Advances in neural information processing systems*, pages 554–560, 2000.

[18] Carl Edward Rasmussen and Zoubin Ghahramani. Occam's razor. In *Advances in neural information processing systems*, pages 294–300, 2001.

[19] Hugo Steinhaus. Sur la division des corp materiels en parties. *Bull. Acad. Polon. Sci*, 1(804): 801, 1956.

[20] Yee Whye Teh. A hierarchical bayesian language model based on pitman-yor processes. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 985–992. Association for Computational Linguistics, 2006.

[21] Yee Whye Teh. Dirichlet process., 2010.

[22] Romain Thibaux and Michael I Jordan. Hierarchical beta processes and the indian buffet process. In *Artificial Intelligence and Statistics*, pages 564–571, 2007.

[23] Junjie Zhang, Qi Wu, Chunhua Shen, Jian Zhang, and Jianfeng Lu. Multilabel image classification with regional latent semantic dependencies. *IEEE Transactions on Multimedia*, 20(10): 2801–2813, 2018.

[24] Yong Zheng, Bamshad Mobasher, and Robin Burke. Context recommendation using multilabel classification. In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 2, pages 288–295. IEEE, 2014.

Appendices

A The Indian buffet process

The Indian buffet process is a metaphor of Indian restaurant which generates a probability distribution on equivalence classes of binary matrices. Assume that the restaurant offers a buffet with infinite number of dishes, and a finite number N of customers enter this restaurant sequentially. Each customer i moves along the buffet and samples each dish k with a probability of m_k/i, where m_k is the number of previous customers who have sampled that dish. Then, the i-th customer tries a number of new dishes, randomly drawn from a Possion(α/i) distribution. The indicator matrix, the (i, j)-th entry showing whether the customer i chose dish j, is a binary matrix from that distribution. It deserves attention that the distribution generated from Indian buffet process is actually exchangeable, although the customers enter in a certain order, because the numbering of the dishes can be arbitrarily permutated.

The algorithm for sampling from the Indian buffet process is provided in Algorithm 2.

Algorithm 2: Sampling scheme from an Indian buffet process

Input: The number of rows N, parameter α
Output: A binary matrix $\mathbf{Z}$

Initialize $\mathbf{Z} = \mathbf{0}_{N\times 0}$, $\boldsymbol{c} = []$, $n = 0$;
for i *in permutated rows of* $\mathbf{Z}$ **do**
  n++;
  for j *in columns of* $\mathbf{Z}$ **do**
    u = sample $\mathbf{Z}[i, j]$ with probability $\boldsymbol{c}[j]/n$;
    if $u == 1$ **then**
      $\boldsymbol{c}[j]$++;
    end
    k = sample from Possion(α/n);
    append $\boldsymbol{c}$ with k 1's;
    right-append $\mathbf{Z}$ with k columns with row i being all 1's;
  end
end

B Auxillary proofs

B.1 Proof of calculating the conditional likelihood

Assume $\mathbf{Z}$ is an $M \times K$ matrix and $M \gg K$. Denote $\lambda = \sigma_X^2/\sigma_A^2$. Let $\mathbf{Z} = \mathbf{U}\mathbf{D}\mathbf{V}^{\mathrm{T}}$ be the singular value decomposition of $\mathbf{Z}$, where $\mathbf{D} = \mathrm{diag}\{d_1, \cdots, d_K\}$.

$$\begin{aligned}\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \lambda\mathbf{I} &= \mathbf{V}\mathbf{D}^2\mathbf{V}^{\mathrm{T}} + \lambda\mathbf{I} \\ &= \mathbf{V}(\mathbf{D}^2 + \lambda\mathbf{I})\mathbf{V}^{\mathrm{T}}.\end{aligned} \tag{19}$$

Therefore, $|\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \lambda\mathbf{I}| = |\mathbf{D}^2 + \lambda\mathbf{I}| = \prod_{k=1}^{K}(d_k^2 + \lambda)$, and

$$\mathrm{tr}\left(\mathbf{X}^{\mathrm{T}}\left(\mathbf{I} - \mathbf{Z}\left(\mathbf{Z}^{\mathrm{T}}\mathbf{Z} + \lambda\mathbf{I}\right)^{-1}\mathbf{Z}^{\mathrm{T}}\right)\mathbf{X}\right) = \mathrm{tr}\left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right) - \mathrm{tr}\left(\mathbf{X}^{\mathrm{T}}\mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^{\mathrm{T}}\mathbf{X}\right), \tag{20}$$

where $\mathbf{\Lambda} = \mathrm{diag}\{d_k^2/(d_k^2+\lambda), k = 1, \cdots, K\}$. The second term can be efficiently calculated.

$$\begin{aligned}\mathrm{tr}(\mathbf{X}^{\mathrm{T}}\mathbf{U}\mathbf{\Lambda}\mathbf{U}^{\mathrm{T}}\mathbf{X}) &= \mathrm{tr}\left(\sum_{k=1}^{K}\frac{d_k^2}{d_k^2+\lambda}\mathbf{X}^{\mathrm{T}}\boldsymbol{u}_{\cdot k}\boldsymbol{u}_{\cdot k}^{\mathrm{T}}\mathbf{X}\right)\\ &= \sum_{k=1}^{K}\frac{d_k^2}{d_k^2+\lambda}\boldsymbol{u}_{\cdot k}^{\mathrm{T}}\mathbf{X}\mathbf{X}^{\mathrm{T}}\boldsymbol{u}_{\cdot k}\\ &= \sum_{k=1}^{K}\frac{d_k^2}{d_k^2+\lambda}||\mathbf{X}^{\mathrm{T}}\boldsymbol{u}_{\cdot k}||_2^2.\end{aligned} \tag{21}$$

B.2 Proof of sampling K_i^{new}

$$\widetilde{\mathbf{W}} = \widetilde{\mathbf{Z}}^{\mathrm{T}}\widetilde{\mathbf{Z}} + \lambda\mathbf{I} = \begin{bmatrix}\mathbf{Z}^{\mathrm{T}}\\ \boldsymbol{e}_i^{\mathrm{T}}\end{bmatrix}[\mathbf{Z} \quad \boldsymbol{e}_i] = \begin{bmatrix}\mathbf{W} & \boldsymbol{z}_{i\cdot}\\ \boldsymbol{z}_{\cdot i}^{\mathrm{T}} & 1+\lambda\end{bmatrix}. \tag{22}$$

By elementary transform of block matrix,

$$\begin{bmatrix}\mathbf{I} & \mathbf{0}\\ -\boldsymbol{z}_{\cdot i}^{\mathrm{T}}\mathbf{W}^{-1} & 1\end{bmatrix}\widetilde{\mathbf{W}}\begin{bmatrix}\mathbf{I} & -\mathbf{W}^{-1}\boldsymbol{z}_{\cdot i}\\ \mathbf{0}^{\mathrm{T}} & 1\end{bmatrix} = \begin{bmatrix}\mathbf{W} & \mathbf{0}\\ \mathbf{0}^{\mathrm{T}} & 1+\lambda-\boldsymbol{z}_{\cdot i}^{\mathrm{T}}\mathbf{W}^{-1}\boldsymbol{z}_{\cdot i}\end{bmatrix}. \tag{23}$$

Note that $\boldsymbol{z}_{\cdot i}^{\mathrm{T}}\mathbf{W}^{-1}\boldsymbol{z}_{\cdot i} = \gamma_{ii}$, and let $\mu = 1+\lambda-\gamma_{ii}$, we have

$$\log|\widetilde{\mathbf{W}}| - \log|\mathbf{W}| = \log\mu. \tag{24}$$

It can be shown as well from (23) that

$$\begin{aligned}\widetilde{\mathbf{W}}^{-1} &= \begin{bmatrix}\mathbf{I} & -\mathbf{W}^{-1}\boldsymbol{z}_{\cdot i}\\ \mathbf{0}^{\mathrm{T}} & 1\end{bmatrix}\begin{bmatrix}\mathbf{W}^{-1} & \mathbf{0}\\ \mathbf{0}^{\mathrm{T}} & \mu^{-1}\end{bmatrix}\begin{bmatrix}\mathbf{I} & \mathbf{0}\\ -\boldsymbol{z}_{\cdot i}^{\mathrm{T}}\mathbf{W}^{-1} & 1\end{bmatrix}\\ &= \begin{bmatrix}\mathbf{W}^{-1}+\mu^{-1}\mathbf{W}^{-1}\boldsymbol{z}_{i\cdot}\boldsymbol{z}_{\cdot i}^{\mathrm{T}}\mathbf{W}^{-1} & -\mu^{-1}\mathbf{W}^{-1}\boldsymbol{z}_{i\cdot}\\ -\mu^{-1}\boldsymbol{z}_{i\cdot}^{\mathrm{T}}\mathbf{W}^{-1} & \mu^{-1}\end{bmatrix}.\end{aligned} \tag{25}$$

Hence,

$$\begin{aligned}\widetilde{\mathbf{\Gamma}} &= [\mathbf{Z} \quad \boldsymbol{e}_i]\,\widetilde{\mathbf{W}}^{-1}\begin{bmatrix}\mathbf{Z}^{\mathrm{T}}\\ \boldsymbol{e}_i^{\mathrm{T}}\end{bmatrix}\\ &= \mathbf{Z}(\mathbf{W}^{-1}+\mu^{-1}\mathbf{W}^{-1}\boldsymbol{z}_{i\cdot}\boldsymbol{z}_{\cdot i}^{\mathrm{T}}\mathbf{W}^{-1})\mathbf{Z}^{\mathrm{T}} - \mu^{-1}\boldsymbol{e}_i\boldsymbol{z}_{i\cdot}^{\mathrm{T}}\mathbf{W}^{-1}\mathbf{Z}^{\mathrm{T}} - \mu^{-1}\mathbf{Z}\mathbf{W}^{-1}\boldsymbol{z}_{i\cdot}\boldsymbol{e}_i^{\mathrm{T}} + \boldsymbol{e}_i\boldsymbol{e}^{\mathrm{T}}\\ &= \mathbf{\Gamma} + \mu^{-1}(\boldsymbol{\gamma}_{\cdot i}\boldsymbol{\gamma}_{\cdot i}^{\mathrm{T}} - \boldsymbol{e}_i\boldsymbol{\gamma}_{\cdot i}^{\mathrm{T}} - \boldsymbol{\gamma}_{\cdot i}\boldsymbol{e}_i^{\mathrm{T}} + \boldsymbol{e}_i\boldsymbol{e}_i^{\mathrm{T}})\\ &= \mathbf{\Gamma} + \mu^{-1}(\boldsymbol{\gamma}_{\cdot i} - \boldsymbol{e}_i)(\boldsymbol{\gamma}_{\cdot i} - \boldsymbol{e}_i)^{\mathrm{T}}.\end{aligned} \tag{26}$$

Therefore, the trace can be updated via

$$\begin{aligned}\widetilde{t} - t &= \mathrm{tr}(\mathbf{X}^{\mathrm{T}}(\mathbf{I}-\widetilde{\mathbf{\Gamma}})\mathbf{X}) - \mathrm{tr}(\mathbf{X}^{\mathrm{T}}(\mathbf{I}-\mathbf{\Gamma})\mathbf{X})\\ &= \mathrm{tr}(\mathbf{X}^{\mathrm{T}}(\mathbf{\Gamma}-\widetilde{\mathbf{\Gamma}})\mathbf{X})\\ &= -\mu^{-1}\mathrm{tr}(\mathbf{X}^{\mathrm{T}}(\boldsymbol{\gamma}_{\cdot i}-\boldsymbol{e}_i)(\boldsymbol{\gamma}_{\cdot i}-\boldsymbol{e}_i)^{\mathrm{T}}\mathbf{X})\\ &= -\mu^{-1}||\mathbf{X}^{\mathrm{T}}(\boldsymbol{\gamma}_{\cdot i}-\boldsymbol{e}_i)||_2^2.\end{aligned} \tag{27}$$

It is worthwhile to notice that to do not need calculating $\mathbf{W}$ and $\mathbf{\Gamma}$, as we do not care about constant terms in posterior probability. What we actually need is the ratio

$$\frac{p(\mathbf{X}\mid\mathbf{Z}_{i,k}^{+},\sigma_X,\sigma_A)P(K_i^{\text{new}}=k\mid\alpha)}{p(\mathbf{X}\mid\mathbf{Z},\sigma_X,\sigma_A)P(K_i^{\text{new}}=0\mid\alpha)} = \prod_{j=0}^{k-1}\frac{p(\mathbf{X}\mid\mathbf{Z}_{i,j+1}^{+},\sigma_X,\sigma_A)P(K_i^{\text{new}}=j+1\mid\alpha)}{p(\mathbf{X}\mid\mathbf{Z}_{i,j}^{+},\sigma_X,\sigma_A)P(K_i^{\text{new}}=j\mid\alpha)}, \tag{28}$$

or equivalently

$$\begin{aligned}&\log p(\mathbf{X}\mid\mathbf{Z}_{i,j+1}^{+},\sigma_X,\sigma_A) - \log p(\mathbf{X}\mid\mathbf{Z}_{i,j}^{+},\sigma_X,\sigma_A)\\ &\qquad + \log P(K_i^{\text{new}}=j+1\mid\alpha) - \log P(K_i^{\text{new}}=j\mid\alpha)\end{aligned} \tag{29}$$

for $j = 0, 1, \cdots, k-1$. In a recursive scheme, we can always view $\mathbf{Z}_{i,j}^{+}$ as baseline and $\mathbf{Z}_{i,j+1}^{+}$ is exactly adding a column $\boldsymbol{e}_i$ on the right of $\mathbf{Z}_{i,j}^{+}$.

Using the notations above, (29) can be evaluated as

$$D \log\left(\frac{\sigma_X}{\sigma_A}\right) - \frac{D}{2}\log\mu + \frac{1}{2\sigma_X^2}\mu^{-1}||\mathbf{X}^{\mathrm{T}}(\boldsymbol{\gamma}_{\cdot i} - \boldsymbol{e}_i)||_2^2. \tag{30}$$

Recall that $\mu = 1 + \lambda - \gamma_{ii}$ which can be obtained from $\boldsymbol{\gamma}_{\cdot i}$. It suffices to show that $\widetilde{\boldsymbol{\gamma}}_{\cdot i}$ can be updated directly through $\boldsymbol{\gamma}_{\cdot i}$. It immediately follows (26) that

$$\widetilde{\boldsymbol{\gamma}}_{\cdot i} = \boldsymbol{\gamma}_{\cdot i} + \mu^{-1}(\gamma_{ii} - 1)(\boldsymbol{\gamma}_{\cdot i} - \boldsymbol{e}_i). \tag{31}$$

C Package Installation

The implemented package, **IBPILFM**[14], can be downloaded and installed via `pip`.

```
$ pip install -i https://test.pypi.org/simple/ IBPILFM
```

To upgrade, run following codes in terminal. (The latest version is `0.1a4` as for Apr 30, 2020.)

```
$ pip install --upgrade -i https://test.pypi.org/simple/ IBPILFM
```

You can check the status of installation via

```
$ python
```

and in python interactive shell, run

```
>>> import ibp
>>> ibp.__version__
```

The package is successfully installed if version is displayed (e.g. `'ibp-0.1a4'`).

D Collaborators

Bo Liu, Master's in Statistical Science, Duke University.

- Author of the package
- Mathematical derivations
- Example on poker cards
- Report writeup

Linlin Li, Master's in Statistical Science, Duke University.

- Co-author of the package
- Unit tests
- Example on Tetris blocks
- Package maintenance