STA663-Final-Report

April 30, 2020

1 Implementation of Indian Buffet Process Prior in an Infinite Latent Feature Model

1.1 By Xiaohe Yang & Zhi Qiu

1.1.1 Github Repository:

https://github.com/final-project-sta663/Indian-Buffet-Process

1.1.2 Package Installation:

pip install -i https://test.pypi.org/simple/ IBP-Gaussian-663-2020==0.0.1

1.1.3 Abstract

Latent feature model is a fundamental concept in machine learning. As the dataset has its own correct classification, however, we never know the number of classes and the classes themselves beforehand when we study the data, so it is more readonable to have a generative process that will automaticlaly create more classes along the process. Infinite latent feature model is also of great importance, as it deploys the case where each feature can belong to multiple classes. Indian Buffect Process(IBP) is a fundamental algorithm in Bayesian nonparametrics, it also models the infinite number of features to reveal latent structure of the data. We explored the paper "Infinite Latent Feature Models and the Indian Buffet Process," which suggests IBP as a prior distribution to use in the latent feature model. In this project, in order to further explore the paper, we implemented the IBP process together with Gibbs Sampler, performed testing on a simulated image dataset, improved the original algorithm by changing structure of likelihood computation, optimized codes using Cython and JIT, and compared the performance with another algorithm that address a similar problem.

1.1.4 Background

In latent class models such as finite mixture models, observed data points are realizations from some distribution determined by a single class [2]. Usually, the number of such classes has to be chosen and fixed. Infinite Dirichlet process mixture models with priors like Chinese Reataurant Process (CRP) break this limitation by allowing an infinite number of latent classes and allow clustering with a potentially infinite number of clusters. However, each point is still limited to one cluster and in real world settings data points are very likely to share multiple classes. For instance, in the problem of clustering instances of human beings appearing in different contexts across a set of images, all such instances to share a cluster for human to different degrees depending on the context in which they appear [2]. Therefore, latent feature models are used to model these problems

where data points share multiple clusters. Similar to the latent class cases, for finite latent feature models the number of features is finite and has to be specified a priori, while in infinite latent feature models such a pre-selecting the number of features could be avoided. Specifically, Indian Buffet Process (IBP) is a quite popular prior distribution that could be used in an infinite latent feature model [1].

There are many sampling models that could be used together with IBP to conduct infinite latent feature modelling. In this paper we will mainly focus on the Gaussian sampling distribution and explore IBP's application in an infinite binary linear-Gaussian latent feature model.

1.1.5 Algorithm Description

- Indian Buffet Process Firstly, IBP is a stochastic process for defining the probability distribution over equivalence classes of sparse binary matrices with a finite number of rows and an unbounded number of columns [6]. It is a metaphor of Indian restaurants offering buffets with a close-to-infinite number of dishes, and the number of dishes (latent features) chosen by a customer follows a Poisson distribution [3].

The detailed procedure is as the following: N customers enter a restaurant one after another. The first customer starts at the left of the buffet and takes a serving from each dish, stopping after a Poisson(α) number of dishes. The ith customer moves along the buffet, sampling dishes in proportion to their popularity (thus customers/obervations are not independent of each other, but note that the dishes/latent features are independent), taking dish k with probability $\frac{m_k}{i}$, where m_k is the number of previous customers who have sampled that dish. Having reached the end of all previous sampled dishes, the ith customer then tries a Poisson($\frac{\alpha}{i}$) number of new dishes. Customer choice of dishes is indicated using a binary matrix Z with N rows and infinitely many columns(corresponding to the infinitely many selection of dishes), where $z_{ik} = 1$ if the ith costumer sampled the kth dish [1].

Mathematically, the probability of a binary matrix, $Z \sim IBP(\alpha)$ is given by

$$P(Z|\alpha) = \frac{\alpha^K}{\prod_{h=1}^{2^N-1} K_h!} exp\{-\alpha H_N\} \prod_{k=1}^{K} \frac{(N-m_k)!(m_k-1)!}{N!}$$

N - Number of objects/obeservations

K - Total number of latent features

K_h - Number of features with history h (whether the N objects possess this feature)

$H_N = \sum_{k=1}^{N} \frac{1}{k}$ - the N^{th} harmonic number

m_k - Number of objects with feature k

α - parameter influencing the Indian Buffet Process's number of features

A detailed derivation of the equation can be found in Griffiths & Ghahramani, 2005 [1].

The conditional distribution in IBP (which is the infinite case) is be given by

$$P(z_{i,k} = 1 \mid z_{-i,k}) = \frac{m_{-i,k}}{N}$$

where $z_{-i,k}$ is the assignment of feature k for all objects except the ith object and $m_{-i,k}$ is the number of objects with feature k except the ith object.

Then, we use IBP as a prior and combine it with a Linear Gaussian sampling model to do Gibbs sampling/ Metropolis Hastings for an infinite binary linear-Gaussian latent feature model.

- Linear-Gaussian Binary Latent Feature Model The mathematical set-up is as the following:

Z - a binary feature ownership matrix (indicator matrix), where $z_{i,k} = 1$ indicating that object i posssess latent feature k

$Z \sim IBP(\alpha)$

X - real-valued observation matrix, where $x_{i,j}$ is the value of feature j for object i, a NxD matrix

$$x_i \sim \text{Normal}(z_i A, \sigma_X^2 I)$$

A - a KxD matrix of weights representing the K latent features.

$$A \sim \text{Normal}(0, \sigma_A^2 I)$$

The likelihood is (after marginalizing out A):

$$P(X|Z, \sigma_X, \sigma_A) = \frac{1}{(2\pi)^{ND/2}(\sigma_X)^{(N-K)D}(\sigma_A)^{KD}(|Z^T Z + \frac{\sigma_X^2}{\sigma_A^2} I|)^{D/2}} exp\{-\frac{1}{2\sigma_X^2} tr(X^T(I - ZMZ^T)X)\}$$

where

$$M = (Z^T Z + \frac{\sigma_X^2}{\sigma_A^2} I)^{-1}$$

- Monte Carlo Simulation Structure We have five parameters of interest in this model that need to be updated throughout the MCMC process.

1.) Z: feature ownernship matrix

2.) K_+: number of new latent features

3.) α: parameter controlling K_+

4.) σ_X

5.) σ_A

There are full conditional distributions for Z, K_+, and α, so can use Gibbs Sampling (a special case of Metropolis Hastings). AS for σ_X and σ_A, the general Metropolis-Hastings algorithm is applied.

Priors:

$$P(z_{ik} = 1|\mathbf{z}_{-i,k}) = \frac{m_{-i,k}}{N}$$

$$\alpha \sim \text{Gamma}(1, 1)$$

$$K_+ \sim \text{Poisson}(\frac{\alpha}{N})$$

Gibbs Sampling Updates:

1. For observation i with more than one feature, sample $z_{i,k}$ using the full conditional distribution.

$$P(z_{ik}|X, Z_{-(i,k),}, \sigma_X, \sigma_A) \propto P(X|Z, \sigma_X, \sigma_A) * P(z_{ik} = 1|\mathbf{z}_{-i,k})$$

2. Sample new features for observation i by first computing a truncated distribution for K_+ using the data likelihood and the prior for K_+ up to 4 new features, and then sampling from the truncated distribution.
3. Sample $P(\alpha \mid Z) \sim Gamma(1 + K_+, 1 + \sum_{i=1}^{N} H_i)$

Metropolis-Hasting Updates:

Update σ_X and σ_A using Metropolis-Hastings. For σ_X, generate a random value from a Uniform(-.05, .05) distribution and add this value to our current value of σ_X to get σ_X^*. Accept new value of σ_X with probability:

$$p = min(1, \frac{P(X|Z, \sigma_X^*, \sigma_A}{P(X|Z, \sigma_X, \sigma_A})$$

To update σ_A, follow the same proceedure as with σ_X, replacing σ_X with σ_A.

1.1.6 Optimization

- Algorithm:

After profiling (output result is stored in Original_Algorithm.ipynb), the likelihood function part is what takes the most of computation time. To reduce the number of calls and speed up sampling, the organization of the sampling functions is modified: the calculation of the M matrix is moved into the likelihood function.

As is shown in the time comparison table for likelihood table in Optimized_Algorithm.ipynb, there is an obvious improvement in time (from 0.000451 to 0.000307) with the new likelihood function. Moreover, as shown in the time comparison table for overall MCMC samling, the optimized algorithm improved from a time of 46.409353 to 41.323922, which is a 10.96% improvement.

- JIT

The JIT decorator is also applied for optimization and the result is stored in Optimized_Algorithm_JIT.ipynb. The time table shows that the original algorithm of a 100 iterations requires a time of 46.486560 while the JIT version only needs 22.399248, achieving an improvement of 51.82%.

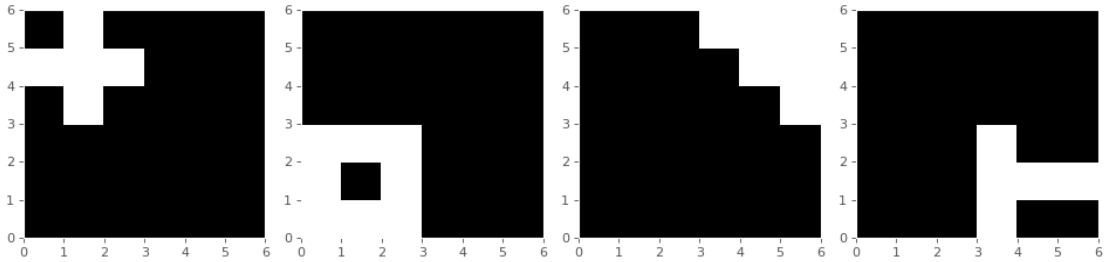
- Cython

As for Cython optimization, we tried to write the MCMC algorithm in C style, the the time comparison table result in Optimized_Cython shows that the time for our original algorithm is 58.640440 while the time of running for the Cython version is 48.795563, which is a 16.79% improvement.

1.1.7 Application to Simulated Data and Real Data

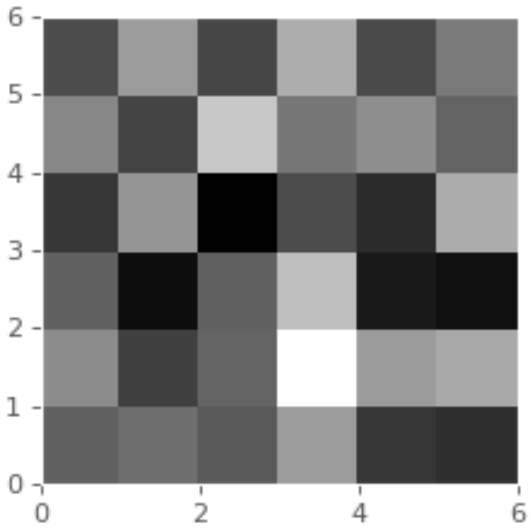
As we faced some problem finding the exact same dataset used in Griffiths and Ghahramani's paper [1], we used the image data proposed by Yildrim [7], which is similar to Griffiths and Ghahramani's data except that the image is of $6x6$ pixels rather than $240x320$ pixels. The number of latent features (objects in image/ image basis) is still 4. That is, in this dataset, four base images were created consisting of $6x6$ pixels and thus each image can be represented by a vector of length 36. The latent features can be represented by a KxD weight matrix A where $K = 4$ and $D = 36$. X is an $N \times D$ matrix that represents the images generated by the K bases (each basis is present with probability 0.5 and $N = 100$).

The four base (latent) images are shown below:

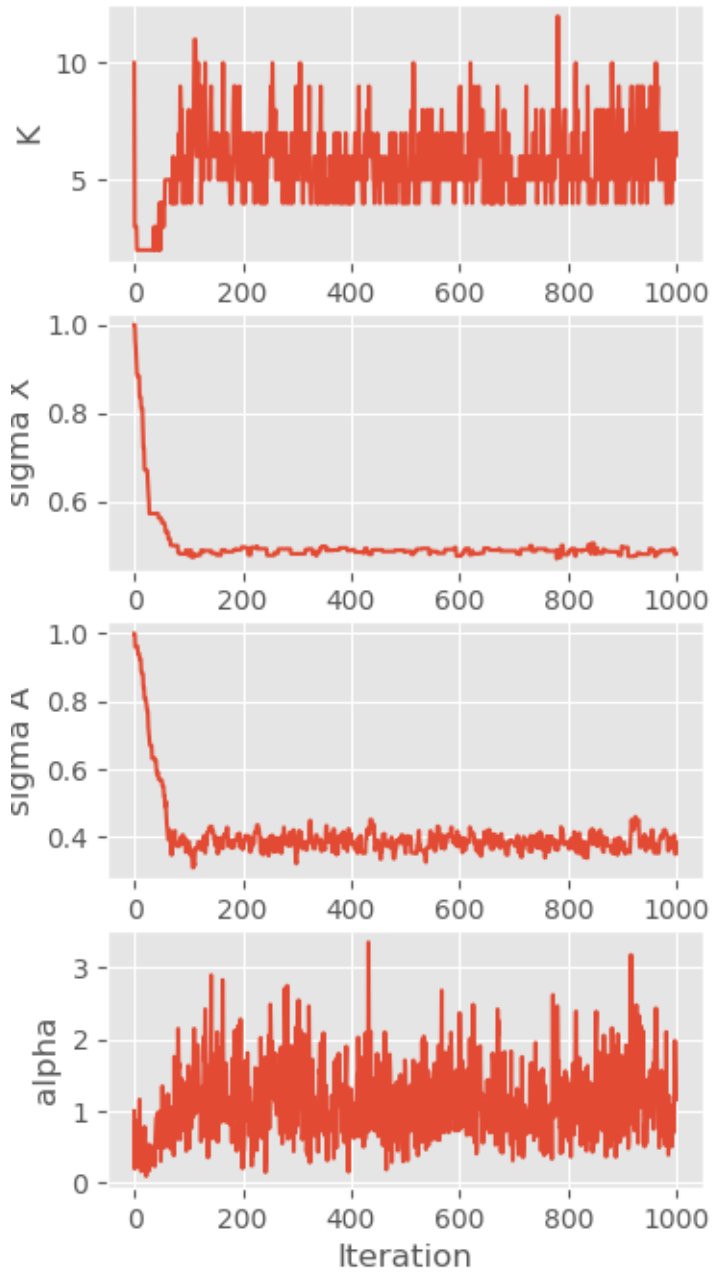


The four bases altogether formulates the latent feature matrices A, then 100 synthetic images are created via simulation, where each image x_i is a superposition of zero or more base images (latent features) with added white noise. X is a $100xD$ matrix. Here, z_i is a row of a binary feature matrix Z of dimension $100xK$. A values in z_i is 1 or 0 with a probability of 0.5. A value of 1 in z_i corresponds to image x_i containing the correspoing base image. σ_x^2, which controls the white noise, is set to 0.5.

An example image of the simulated dataset which is generated by our sampling algorithm is shown as below:



As for the MCMC results from a simulted data set, we represent them by presenting traceplots of the parameters K, σ_X, σ_A and α. It is shown in the above traceplots that the convergence of our Markov chain is good and σ_X converges to its true value 0.5:

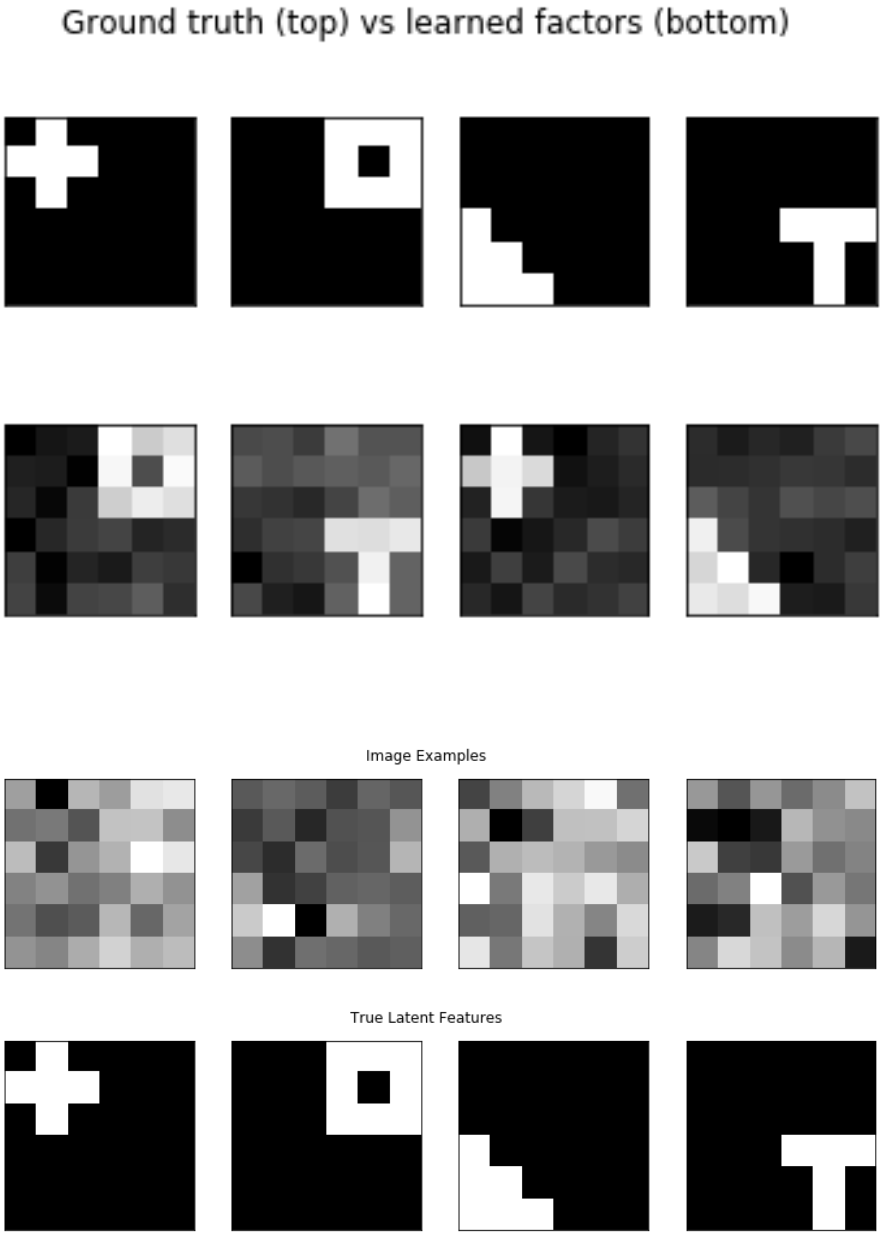


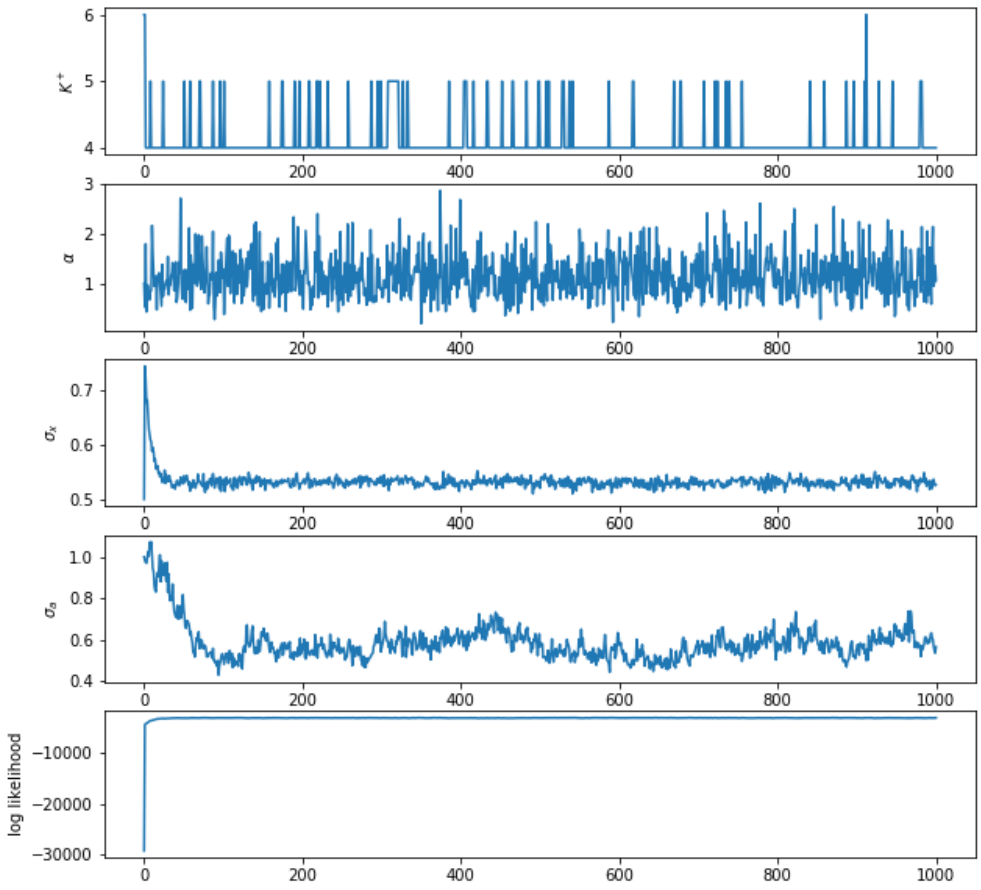
1.1.8 Comparison with Competing Algorithms

We made a comparison of our algorithm and Joo's algorithm, whose package could be retrived on Github [8]. The exacution of his algorithm could be found in the Competing_Algorithm folder and

only the results are included here.

His test simulation results are shown graphically below, which used 100 $6x6$ images with latent feactures of number $K = 6$ and 1000 MCMC iterations.





Comparing the traceplots from his simulation dataset and ours above, σ_x converges faster in his algorithm than ours while σ_A converges slightly faster in our algorithm than his. Both algorithms achieve good mixing.

1.1.9 Conclusion

Overall, we implemented the Indian Buffet Process in python and applied it as a prior in the Infinite linear-Gaussian binary feature model as was shown by Griffiths and Ghahramani [1] and reproduced the data set used in Yildirim [7]. The plain python codes were firstly improved by tructrural change, and then optimized by high performance computing tools such as Cython and JIT. Future work about code optimization could be done in matrix inversion [1] and parallization. Moreover, considering the caracteristic of gibbs sampling that it is a stochastic algorithm, key dactors like initialization, burn-in and lag could be chosen or adjusted more sensibly for practical usage.

1.1.10 References

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

[2] Eric P. Xing. 21: The Indian Buffet Process [Lecture]. 2014.

[3] Christine Chai. Implementation of the Indian Buffet Process. 2015.

[4] Dipesh Gautam. Indian Buffet Process and its application in the Infinite Latent Feature Model. 2015.

[5] Radhika Anand. Infinite Latent Feature Models and the Indian Buffet Process. 2015.

[6] Drew Jordan and Sunith Suresh. The Indian Buffet Process and Applications for Unsupervised Learning. 2016.

[7] Ilker Yildirim. Bayesian statistics: Indian buffet process. 2012.

[8] JaeHyun Joo. A Linear-Gaussian Latent Factor (Feature) Model using an IBP prior [Github]. 2018. https://github.com/jaehyunjoo/IBP_Linear_Gaussian_Latent_Factor_Model