STA663 Final Project
Indian Buffet Process and its application in the Infinite Latent Feature Model

Dipesh Gautam

April 29, 2015

1 Background

1.1 Indian Buffet Process (IBP)

The Indian Buffet is an adaptation of Chinese Buffett Process where each object instead of being associated with a single latent class can be associated with multiple classes. This is particularly useful when each object has multile latent features and by associating objects with a single class we cannot partition them into homogeneous subsets.

In the Indian buffet process, N customers enter a restaurant one after another. Each customer encounters a buffet consisting of infinitely many dishes arranged in a line. 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, 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. Which costumer chose which dishes is indicated using a binary matrix $\mathbf{Z}$ with N rows and infinitely many columns(corresponding to the infinitely many selection of dished), where $z_{ik} = 1$ if the ith costumer sampled kth dish.

2 Implementation

IBP can be used as a prior in models for unsupervised learning. An example of which is presentd in Griffiths and Ghahramani(2005) and Yildirim(2012), where IBP is used as a prior in infinite linear-gaussian model.

2.1 Infinite Latent Features Model

We used the linear-gaussian feature model as derived in Griffiths and Ghahramani(2005). We consider a binary feature ownership matrix Z which represents the presence or absence of the underlying features of the observations X. Thus each D-dimensional object x_i is generated from a Gaussian distribution:

$$x_i \sim Normal(z_i A, \Sigma_X)$$

Where A is KXD matrix of weight representing the K latent features and $\Sigma_X = \sigma_x^2 I$ is the covariance matrix that introduces white noise to the images. The weight matrix A itself has a prior with mean 0, and covariance $\sigma_A^2 I$. Thus, the likelihood is given for the data is given by:

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

2.2 Algorithm for Gibbs sampling and Metropolis-Hastings

2.2.1 Parameters

The parameters of interest are:

- Z: feature ownership matrix
- K_{new}: Number of new features
- α: parameter K_{new}
- σ_X
- σ_A

Full conditionals can be obtained for Z, K_{new} and $alpha$, so, we can update them using Gibbs sampling. For σ_X and σ_A, we'll use random walk MH algorithm for updating.

2.2.2 Priors for Z, K_{new} and α

Gamma prior is set for α

$$\alpha \sim Gamma(1,1) \tag{2}$$

Prior on $\mathbf{Z}$ is obtained by IBP as:

$$P(z_{ik}=1|\mathbf{z}_{-i,k}) = \frac{n_{-i,k}}{N} \tag{3}$$

K_{new} has a poisson prior:

$$K_{new} \sim Poisson(\frac{\alpha}{N}) \tag{4}$$

2.2.3 Full conditional posteriors on Z, K_{new} and α

Full conditional posterior for Z can be directly computed using Equations (1) and (3) as given below:

$$P(z_{ik}|X, Z_{-(i,k),}, \sigma_X, \sigma_A) \propto P(X|Z,\sigma_X,\sigma_A) * P(z_{ik}=1|\mathbf{z}_{-i,k}) \tag{5}$$

Full conditional posterior for α is given by:

$$P(\alpha|Z) \sim Gamma(1+K_+, 1+\sum_{i=1}^{N} H_i) \tag{6}$$

To sample the number of new features for observation i, we use a truncated distribution, computing probabilities for a range of values $K_{new}^{(i)}$ up to an upper bound (say 3). Using the ikelihood and prior given by Equations (1) and (4) respectively, we can easily calculate the probability distribution for K_{new} and sample the number of new dishes accordingly.

2.2.4 Metropolis-Hastings for σ_X and σ_A

For σ_X we use random-walk MH algorithm as follows:

$$\epsilon \sim Uniform(-.05,.05) \tag{7}$$

$$\sigma_X^* = \sigma_X + \epsilon \tag{8}$$

$$\tag{9}$$

Accept this new σ_X with probability given by:

$$AR = min\{1, \frac{P(X|Z,\sigma_X^*,\sigma_A)}{P(X|Z,\sigma_X,\sigma_A)}\} \tag{10}$$

Where AR is the acceptance ratio.
We use similar algorithm to sample σ_A where we replace σ_X by σ_A in the algorithm described above.

3 Unit Testing

4 Profiling and Optimization

We profiled the code using *cProfile* to figure out bottlenecks. The result is shown in *Profiling result*. We see that most of the computational time is spent on calculating the *log likelihood(ll)* and matrix inversion. Due to this fact, one of the first things we looked at were ways to reduce computation time for likelihood and/or inverse calculation.

Profiling Result

```
        2075808 function calls in 10.873 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:linenofunction
   154080    3.985    0.000    3.985    0.000 {method 'dot' of 'numpy.ndarray' objects}
    30816    1.540    0.000    9.411    0.000 <ipython-input-144-816e3f6a3e53>:2ll
    30816    0.770    0.000    1.353    0.000 linalg.py:455inv
        1    0.726    0.726   10.873   10.873 <ipython-input-145-4efe9a6e9287>:1sampler
    30816    0.542    0.000    1.071    0.000 linalg.py:1679det
    61632    0.382    0.000    0.963    0.000 numeric.py:2125identity
    61632    0.367    0.000    0.580    0.000 twodim_base.py:190eye
    30816    0.332    0.000    0.332    0.000 {method 'trace' of 'numpy.ndarray' objects}
    86971    0.267    0.000    0.267    0.000 {numpy.core.multiarray.zeros}
    61632    0.183    0.000    0.314    0.000 linalg.py:139_commonType
   122449    0.171    0.000    0.171    0.000 {numpy.core.multiarray.array}
```

4.1 Matrix Inversion

We tried the matrix inversion method described in Griffiths and Ghahramani(2005, eq 51-54), where the method reduces the runtime by allowing us to perform rank one updates instead when only one value is changed. We implemented the algorithm and were able to speed up the process as shown in Table 1.

Table 1: Comparision of matrix inverse methods

	Time
linalg.inverse	0.000080
calcInverse	0.000038

Even though we were able to improve the performance, due to some numerical errors, we were not able to obtain a stationary MCMC chain using this method. This could be achieved by spending some more time on it but due to time constraints we decided that to look at fixing this at a later time.

4.2 Likelihood function

While working on the optimized matrix inversion, we noticed that the matrix that we're inverting i.e. $(Z^T Z + \frac{\sigma_X^2}{\sigma_A^2} I)$ actually appears twice inside the likelihood function. So, we looked at removing the redundancy by calculating the matrix and storing it. We were able to gain some improvement using this method as shown in Table 2 and Table 3. Since the likelihood function is called numerous times, even the small gain shwon in Table 2 was translated into a substantial gain as shown in Table 3.

Table 2: Runtime Comparision

	Time
original ll function	0.000371
Proposed ll function	0.000324

4.3 Cython

Another way we looked at improving the performance of the code was by cythonizing the code. We again looked at improving the performance of the likelihood function by cythonizing it. As shwon in Table 3, we were not able to gain substantial improvements from it.

4.4 Parallelization and CUDA

Since our algorithm is an MCMC algorithm with serial dependence, parallelization does not seem to be a good idea. One of the ways, parallelization can be done is by splitting the chain into multiple smaller chains and combining them back. We tested it and it showed some improvement in the code but decided against using it as the gain wasn't significant enough as we had to take care of multiple burn-in periods and ignore the loss of markov property due to multiple chains. Also, parallelizing the density calculation for likelihood wasn't an option for our algorithm as we had a discrete density with just two points.

Table 3: Runtime Comparision

	Total Time
Initial Code	539.818143
Improved ll	489.207815
Cythonized	507.522965

5 Application and Results

5.1 Simulated Data

We used simulated data similar to Griffiths and Ghahramani(2005) and Yildirim (2012). The data set consists of $100(N)$ images X, where each image x_i a vector of length $36(D)$ representing the $6X6$ dimension of each object. These images were created using $4(K)$ latent features (base images) which correspond to the rows of the weight matrix A. Each object has .5 probability for presence of each feature. Then the object was created by adding white noise corresponding to $\sigma_X^2 = 0.5$ as:

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

The basis image and first four simulated images are shown in Fig. 3, where the top row are the features, bottom row are the first four objects and the middle row represents presence or absence of each of the four features.

5.2 Results

5.2.1 Validation and K_+

We ran our sampler with the improved likelihood calculation for 1000 iterations. Our data was simulated with 4 latent features. The distribution of K_+(total features detected at each iteration) is shown in Fig.

1(a). Although we see that the mode of the number of features is 5, with significant iterations where the number of features is 7, we can see from Fig. 1(b) that most of the objects actually had between 1 and 4 features with very few of them with 5 or more features. These outliers can be attributed to the noise and variance in the data simulation and ignored. So, we can conclude that the algorithm correctly predicts and detects the existence of 4 latent features in the simulated data.
Trace plot for σ_X, σ_A, and α is shown in Fig. 2. We see that σ_X is converging to it's true value of 0.5 and the other parameters also show proper convergence validating the authenticity of the algorithm.

Figure 1: Distribution of Kplus and mean number of features per object

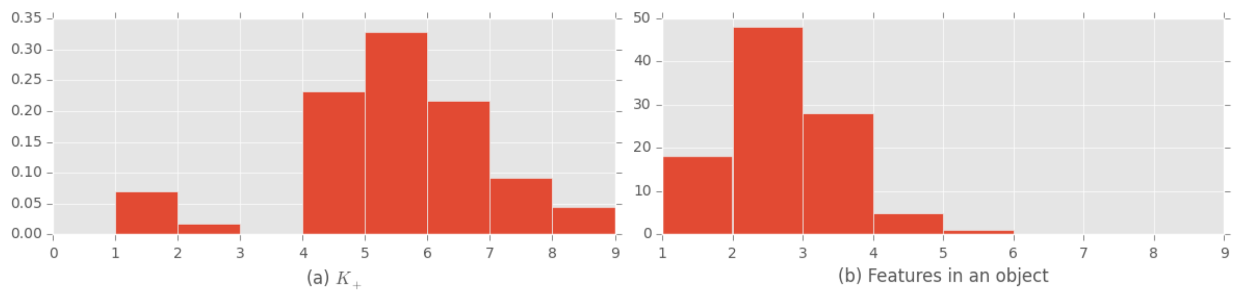
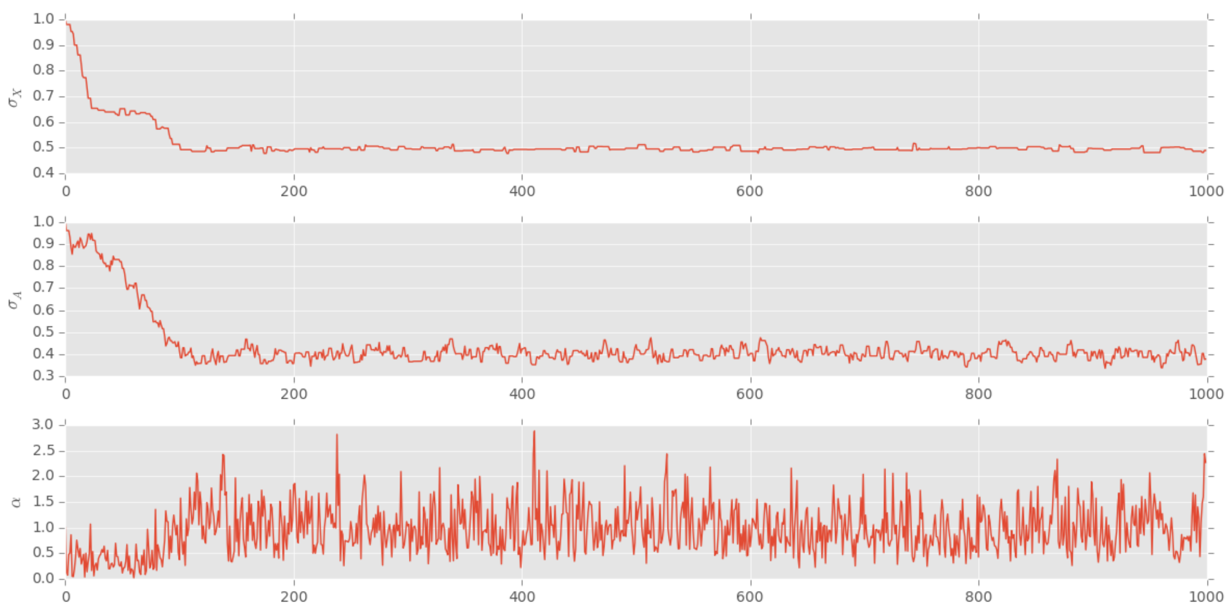


Figure 2: Traceplots for σ_X, σ_A and α



5.2.2 Latent features and recreating objects

Posterior estimation of A is given by:

$$E[A|X,Z] = (Z^T Z + \frac{\sigma_X^2}{\sigma_A^2} I)^{-1} Z^T X$$

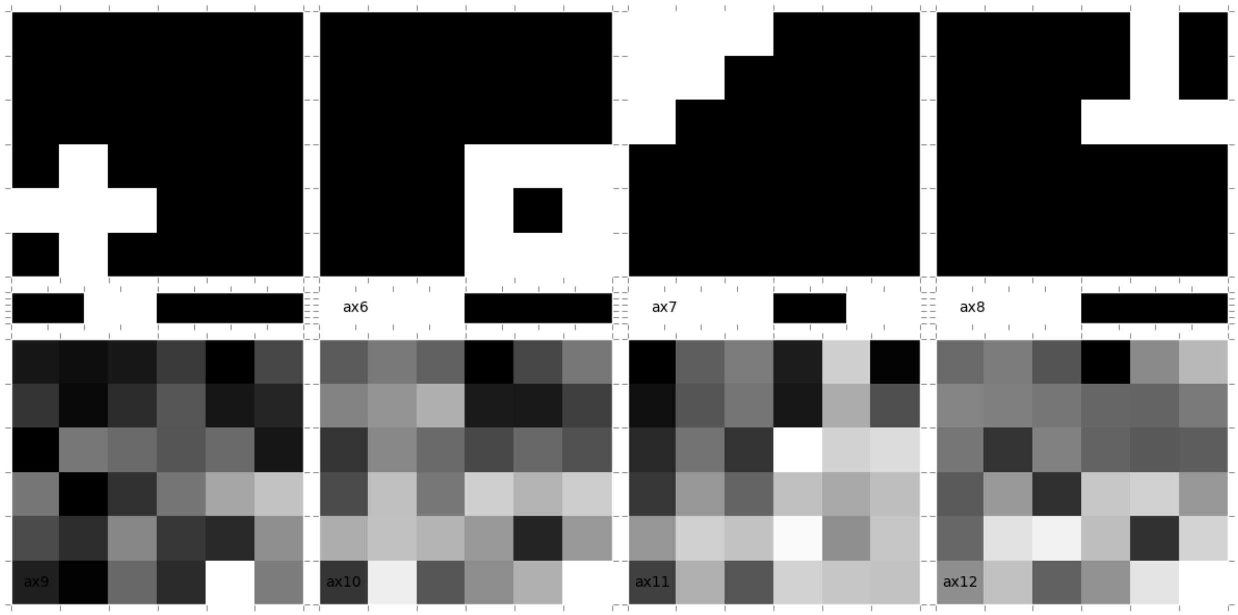


Figure 3: Original Features, First four simulated objects and the features present in each of the objects. First row shows the 4 latent features used to simulate the data. Third row shows the first four simulated objects and the middle row shows the presense or absence of the the latent features (in the order specified in the first row) in the corresponding objects below. Light signifies presence and dark signifies the absence of the feature for any given object.

as givin in Griffiths and Ghahramani(2005) eq.59. For this calculation, we used the final Z obtained from the MC using only the first four columns corresponding to the four detected features. Likewise, posterior mean of σ_X and σ_A was used in the calculation of posterior expectation of weight matrix A.
With this information and the posterior Z, we were able to recreate the objects X as:

$$Xi \sim Normal(ZiA, 0)$$

We used zero variance to ignore the white noise in the recreated images. The results are shown in Fig. 4. By comparing with the original features and simulated objects as given in Fig. 3 with the detected features and recreated objects as given in Fig. 4, we can conclude that the algorithm was successfull in identifying all the latent features and successfully detecting the presense or absence of those features in the simulated objects which had white noise making detection difficult.

6 Comparision

7 Conclusions

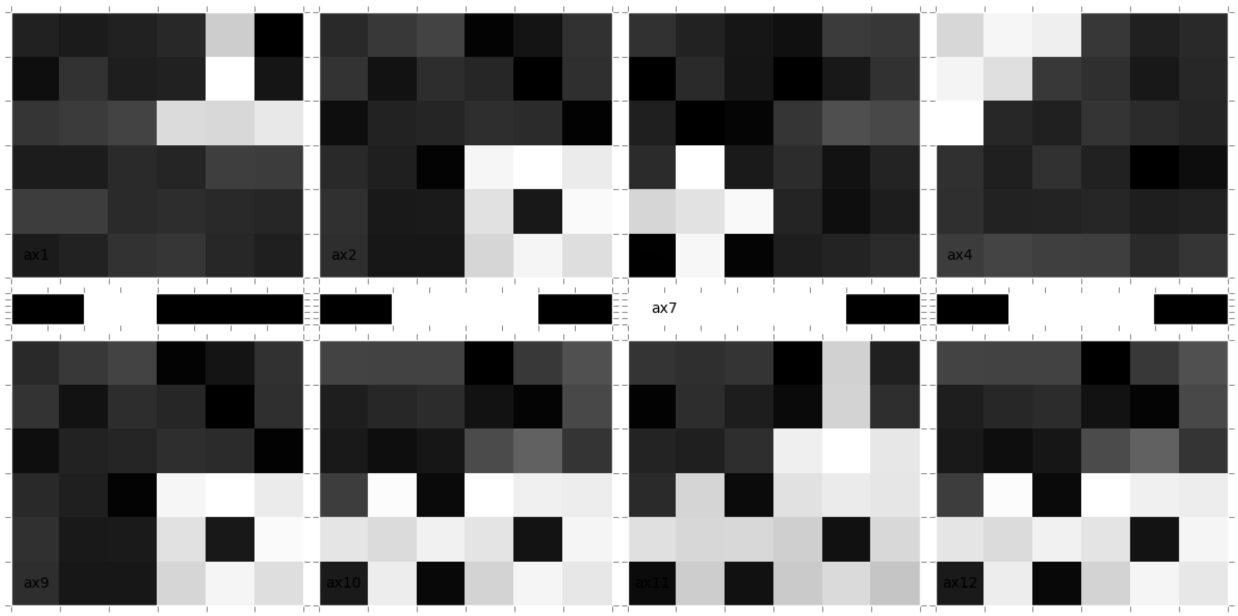


Figure 4: Detected Features, First four recreated objects and the features present in each of the objects. First row shows the 4 latent features used detected by MCMC. Third row shows the first four recreated objects and the middle row shows the presense or absence of the the latent features (in the order specified in the first row) in the corresponding objects below. Light signifies presence and dark signifies the absence of the feature for any given object.