CS281 Section 9: Graph Models and Practical MCMC

Scott Linderman

November 11, 2013

Now that we have a few MCMC inference algorithms in our toolbox, let's try them out on some random graph models. Graphs turn up all over the place. We use them to model social networks, biological systems, and transportation networks, to name a few. A large class of random graph models can be cast in a simple generative framework that allows for natural Gibbs sampling algorithms. We'll derive Gibbs updates for a few models and then test our MCMC implementation using a few standard diagnostics.

The corresponding code for this section is located at `https://github.com/slinderman/cs281sec09`. Clone the code and run the following commands to get started:

```
> python
Python 2.7.2 EPD_free 7.2-2 (32-bit) ...
>>> import demo
```

1. Random Graphs

   We will represent a graph with N nodes by its binary *adjacency matrix* $\boldsymbol{A} \in \{0,1\}^{N\times N}$. The entry $A_{n,n'}$ indicates whether or not a directed edge exists from node n to node n'. It is easy to see that undirected graphs correspond to symmetric adjacency matrices. We seek a distribution over graphs of the form

   $$p(\boldsymbol{A}\,|\,\theta) \equiv p(\{\{A_{n,n'}\}_{n=1}^{N}\}_{n'=1}^{N}\,|\,\theta). \tag{1}$$

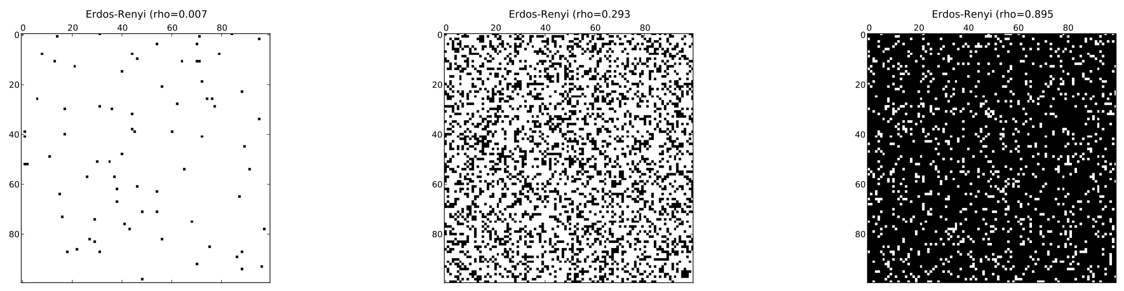
   Given such a distribution we could answer a number of interesting queries.

   One important task is **link prediction**. Suppose you have seen a subset of edges in a social network. Call this subset $\mathcal{A} = \{(n_{1,m}, n_{2,m})\}_{m=1}^{M}$. If I ask you how likely it is than an unobserved edge (n, n') exists, you can answer this with the sum and product rule:

   $$p(A_{n,n'}\,|\,\mathcal{A}) = \int_{\Theta} p(A_{n,n'}, \theta\,|\,\mathcal{A})\mathrm{d}\theta \tag{2}$$

   $$= \int_{\Theta} p(A_{n,n'}\,|\,\theta, \mathcal{A})p(\theta\,|\,\mathcal{A})\mathrm{d}\theta \tag{3}$$

   If we can efficiently compute the first term in this integral and we also have a method of efficiently sampling from the posterior distribution $p(\theta\,|\,\mathcal{A})$ then this integral may be approximated using Monte Carlo. For example, if the entries in $\boldsymbol{A}$ are conditionally independent given θ, this becomes particularly easy.



(a) $\rho = 0.007$ (b) $\rho = 0.293$ (c) $\rho = 0.895$

Figure 1: Erdös-Renyi random graphs.

Another task is to infer something about the latent structure of the graph. We would like a model of adjacency matrices that reflects our inductive biases about how networks are structured. Often these models have interpretable parameters that shed some insight into structure of the graph.

Question: what properties might we wish to express in a random graph model?

(a) Erdös-Renyi Model

One simple property is **sparsity**. Many real-world graphs contain only a fraction of all possible edges. This motivates a simple probabilistic model in which the entries are drawn from a distribution biased toward zero[1]. Perhaps the simplest such model is the **Erdös-Renyi** random graph. In this model,

$$\theta = \{\rho\}, \tag{4}$$

$$\rho \sim \text{Beta}(\rho \mid \beta_1, \beta_0) \tag{5}$$

$$A_{n,n'} \mid \rho \sim_{i.i.d.} \text{Bern}(\rho). \tag{6}$$

Each entry is an independent and identically distributed Bernoulli random variable. Given the beta-Bernoulli conjugacy, we can easily calculate the posterior distribution over ρ.

Question: what are the parameters of the posterior distribution over ρ?

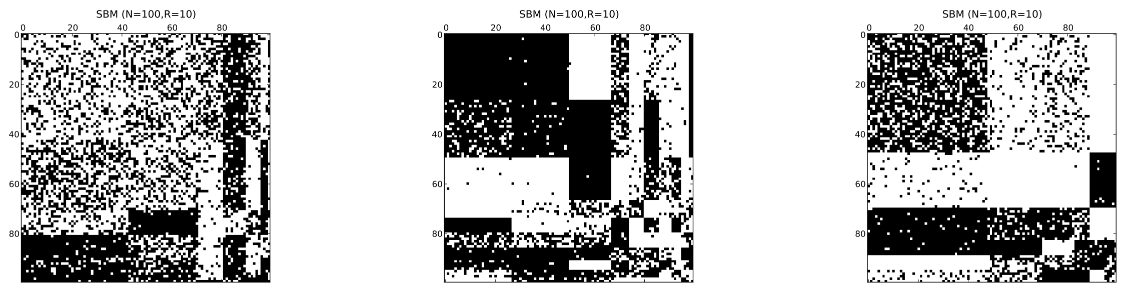
You can generate a few samples of Erdös-Renyi random graphs, as in Figure**??**, by running the following command:

```
>>> demo.sample_er()
```

(b) Block Models

Another common facet of real-world networks is community structure. Nodes often cluster together into densely connected packs. More generally, the probability of an edge between two nodes may be a function of the latent clusters or "blocks" to which

[1]Technically, a graph is typically considered sparse if the expected degree of any node is finite as $N \to \infty$



(a) $\beta_1 = \beta_0 = 1.0$, $\alpha = 0.5$ (b) $\beta_1 = \beta_0 = 0.3$, $\alpha = 1$ (c) $\beta_1 = \beta_0 = 0.3$, $\alpha = 0.5$

Figure 2: Stochastic block models with $N = 100$, $R = 10$.

the nodes belong. This motivates the **stochastic block model**. A generative model for an SBM with a fixed number of blocks R is:

$$\theta = \left\{ \begin{array}{l} \boldsymbol{B} \in [0,1]^{R\times R}, \\ \boldsymbol{\pi} \in [0,1]^{R} \text{ s.t. } \sum_r \pi_r = 1, \\ \boldsymbol{z} \in \{1, \dots, R\}^N \end{array} \right\} \tag{7}$$

$$B_{r,r'} \sim_{i.i.d.} \text{Beta}(B_{r,r'} \mid \beta_1, \beta_0) \tag{8}$$

$$\boldsymbol{\pi} \sim \text{Dir}(\alpha) \tag{9}$$

$$z_n \mid \boldsymbol{\pi} \sim_{i.i.d.} \text{Cat}(\boldsymbol{\pi}) \tag{10}$$

$$A_{n,n'} \mid \boldsymbol{z}, \boldsymbol{B}, \pi \sim_{i.i.d.} \text{Bern}(B_{z_n, z_{n'}}). \tag{11}$$

Write down the joint distribution $p(\boldsymbol{A}, \boldsymbol{z}, \boldsymbol{B}, \boldsymbol{\pi})$**.**

Samples from this model, with varying β's and α's, are shown in Figure **??**.

You can reproduce this figure by running the following command:

```
>>> demo.sample_sbm()
```

(c) Aldous-Hoover Representation It turns out that the stochastic block model and the Erdös-Renyi model are special cases of the general family of **exchangeable** random graphs. That is, the probabiliy of the graph would not change if we permuted the labels of the nodes. This is intuitively akin to reordering rows and columns of the adjacency matrix (making sure to leave the edge structure intact).

It turns out that graphs with this property have a simple generative model in which the entries in $\boldsymbol{A}$ are conditionally independent given some features of the nodes. In the Erdös-Renyi case there are no features, each entry is an i.i.d. Bernoulli. In the stochastic block model, given the block assignments the edges are conditionally independent Bernoullis. Other models in this class include mixed membership block models, latent distance models, latent eigenmodels, and nonparametric extensions of the above. We won't go into more detail on this subject, but this theory will tie into the Bayesian nonparametric models we will discuss later in the course.

2. MCMC Inference

Our goal is to infer the latent variables of the graph model given some observations of the edge. In the SBM we seek to infer the block assignments $\boldsymbol{z}$ and the parameters of the distribution $\boldsymbol{B}$ and $\boldsymbol{\pi}$. In this section we will make use of Gibbs sampling[2].

Tasks:

- **Write pseudocode for the Gibbs sampling algorithm.**
- **Derive the Gibbs updates for B, and π.**

Once we have found the posterior distribution for the parameters with conjugate priors, we must also derive the conditional distribution for $z_n \mid z_{\neg n}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi}$. To do so, we return to our joint distribution.

Show that the conditional distribution is proportional to the joint distribution.

All we have to do is collect the terms that have z_n's in them. (This is $p(\boldsymbol{A} \mid \boldsymbol{z})$ and $p(\boldsymbol{z} \mid \boldsymbol{\pi})$).

Derive the unnormalized probabiliy $\tilde{p}(z_n \mid z_{\neg n}, \boldsymbol{A}, \boldsymbol{\pi})$.

When you implement this you will sample from this conditional distribution using the log-sum-exp trick to avoid numerical overflow.

To run this Gibbs sampling algorithm on randomly generated stochastic block models, run the following command. It will show the true network on the left and the current iteration's inferred block structure on the right. Once the Gibbs sampler has finished it will display a trace of the log probability at each iteration.

```
>>> demo.gibbs_sbm()
```

(a) MCMC Diagnostics

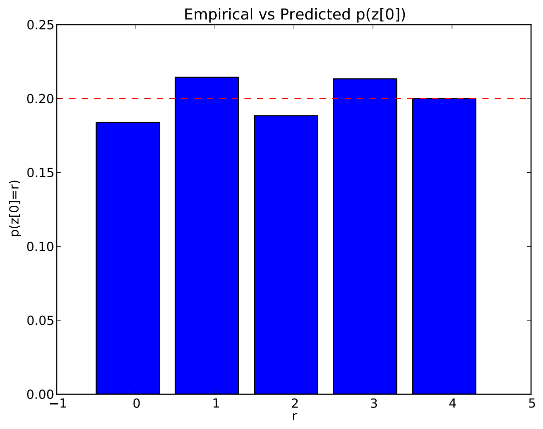
How do you know when your algorithm is working? Well, a first start is to see whether it can recover ground truth on data drawn from the model.

i. Does the log probability plateau or is it steadily climbing? The latter indicates that it has not burned in.
ii. Do multiple Markov chains converge to the same values or log probability? If not, you're probably getting stuck in local maxima.
iii. See R-CODA and PyMC for other diagnostic tools (effective sample size, etc.)

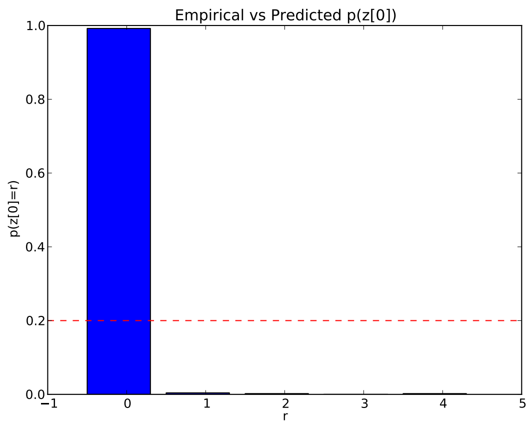
How do you know if you have a bug in your Gibbs updates? **Geweke validation** is a simple way to test if you have a generative model for the data. The idea is simple: add a step in your Gibbs sampler that samples $\boldsymbol{A} \mid \theta$. Then the stationary distribution of your Markov chain is the joint distribution. We can easily get marginal distributions for the parameters and compare them to their expected marginals (i.e. the prior for root-level variables). Typically you only need to generate a small dataset in order to expose bugs.

To see an example of a good Geweke result, run

[2]Though it is debatable whether this is the best approach for this model since there tend to many peaks in the posterior.



(a) Desired $p(z_0)$



(b) Erroneous $p(z_0)$

Figure 3: Geweke validation of $p(z_0)$ for working code and code with a bug.

```
>>> demo.geweke_sbm_test()
```

Let's look at a couple failure cases due to bugs in our Gibbs sampler for the SBM. We sampled an SBM with $N = 20$ nodes and $R = 5$ blocks for 5000 iterations. We used a symmetric Dirichlet prior α so we expect that the marginal distribution over block assignments for any node should be uniform. Figure **??** shows the results when the code is working and when we have a bug. In particular, the bug was an indexing error which caused us to almost always assign nodes to the first block.

To reproduce the failures in Figure **??**, run

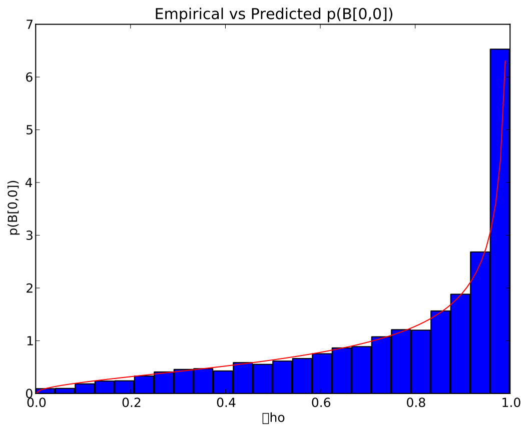
```
>>> demo.geweke_sbm_test(bug1=True)
```

If the Gibbs sampler were working correctly, we would have expected the empirical distribution to match the red line, as in Figure **??**. However, when there is a bug in the code, the empirical distribution will often be heavily skewed, as in'Figure **??**.

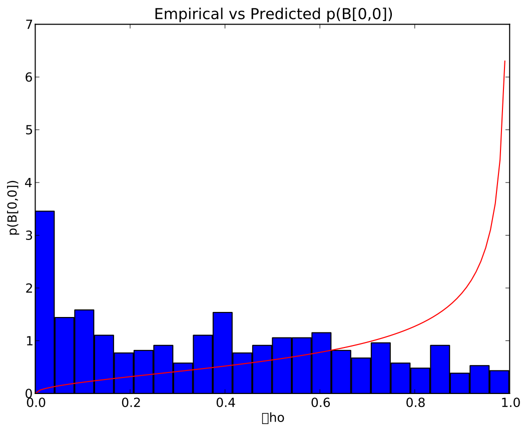
We can do the same for the parameter $\boldsymbol{B}$. The marginal distribution over $\boldsymbol{B}$'s entries is $\text{Beta}(\beta_1, \beta_0)$. As we see in Figure **??**, after 5000 iterations the empirical and predicted distributions match very well. However, if we accidentally swap the parameters of the beta distribution in our Gibbs update we see dramatically different results, as in Figure **??**.

To reproduce the failures in Figure **??**, run

```
>>> demo.geweke_sbm_test(bug3=True)
```



(a) Desired $p(B_{1,1})$



(b) Erroneous $p(B_{1,1})$

Figure 4: Geweke validation of $p(B_{1,1})$ for working code and code with a bug.