

Biostatistics 615/815 Lecture 18: Gibbs Sampling

Hyun Min Kang

November 15th, 2011

Recap: The E-M algorithm

Expectation step (E-step)

- Given the current estimates of parameters $\lambda^{(t)}$, calculate the conditional distribution of latent variable \mathbf{z} .
- Then the expected log-likelihood of data given the conditional distribution of \mathbf{z} can be obtained

$$Q(\lambda|\lambda^{(t)}) = \mathbf{E}_{\mathbf{z}|\mathbf{x},\lambda^{(t)}} [\log p(\mathbf{x}, \mathbf{z}|\lambda)]$$

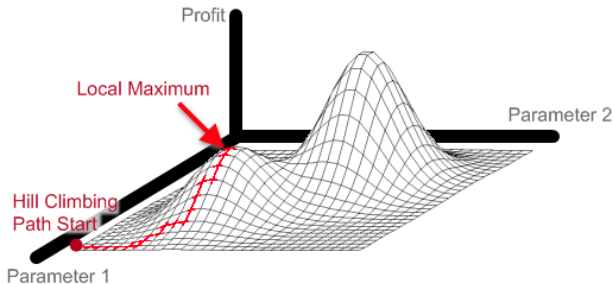
Maximization step (M-step)

- Find the parameter that maximize the expected log-likelihood

$$\lambda^{(t+1)} = \arg \max_{\lambda} Q(\lambda|\lambda^t)$$

Recap - Local minimization methods

The problem with hill climbing is that it gets stuck on "local-maxima"

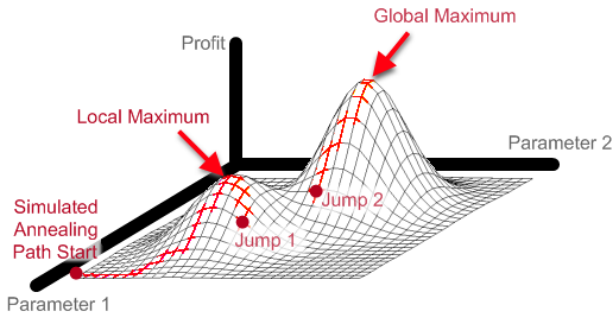


Images by Max Dama from

<http://maxdama.blogspot.com/2008/07/trading-optimization-simulated.html>

Recap - Global minimization with Simulated Annealing

Simulated Annealing can escape local minima with chaotic jumps



Images by Max Dama from

<http://maxdama.blogspot.com/2008/07/trading-optimization-simulated.html>

Recap - Simulated Annealing Recipes

- 1 Select starting temperature and initial parameter values
- 2 Randomly select a new point in the neighborhood of the original
- 3 Compare the two points using the *Metropolis criterion*
- 4 Repeat steps 2 and 3 until system reaches equilibrium state
 - In practice, repeat the process N times for large N .
- 5 Decrease temperature and repeat the above steps, stop when system reaches frozen state

Optimization Strategies

- Single Dimension
 - Golden Search
 - Parabolic Approximations
- Multiple Dimensions
 - Simplex Method
 - E-M Algorithm
 - Simulated Annealing

Optimization Strategies

- Single Dimension
 - Golden Search
 - Parabolic Approximations
- Multiple Dimensions
 - Simplex Method
 - E-M Algorithm
 - Simulated Annealing
 - Gibbs Sampling

Gibbs Sampler

- Another MCMC Method
- Update a single parameter at a time
- Sample from conditional distribution when other parameters are fixed

Gibbs Sampler Algorithm

- 1 Consider a particular choice of parameter values $\lambda^{(t)}$.
- 2 Define the next set of parameter values by
 - Selecting a component to update, say i .
 - Sample value for $\lambda_i^{(t+1)}$, from $p(\lambda_i|x, \lambda_1, \dots, \lambda_{i-1}, \lambda_{i+1}, \dots, \lambda_k)$.
- 3 Increment t and repeat the previous steps.

An alternative Gibbs Sampler Algorithm

- 1 Consider a particular choice of parameter values $\lambda^{(t)}$.
- 2 Define the next set of parameter values by
 - Sample value for $\lambda_1^{(t+1)}$, from $p(\lambda_1|x, \lambda_2, \lambda_3, \dots, \lambda_k)$.
 - Sample value for $\lambda_2^{(t+1)}$, from $p(\lambda_2|x, \lambda_1, \lambda_3, \dots, \lambda_k)$.
 - ...
 - Sample value for $\lambda_k^{(t+1)}$, from $p(\lambda_k|x, \lambda_1, \lambda_2, \dots, \lambda_{k-1})$.
- 3 Increment t and repeat the previous steps.

Gibbs Sampling for Gaussian Mixture

Using conditional distributions

- Observed data : $\mathbf{x} = (x_1, \dots, x_n)$
- Parameters : $\lambda = (\pi_1, \dots, \pi_k, \mu_1, \dots, \mu_k, \sigma_1^2, \dots, \sigma_k^2)$.
- Sample each λ_i from conditional distribution - not very straightforward

Using source of each observations

- Observed data : $\mathbf{x} = (x_1, \dots, x_n)$
- Parameters : $\mathbf{z} = (z_1, \dots, z_n)$ where $z_i \in \{1, \dots, k\}$.
- Sample each z_i conditioned by all the other \mathbf{z} .

Update procedure in Gibbs sampler

$$\Pr(z_j = i | x_j, \lambda) = \frac{\pi_i \mathcal{N}(x_j | \mu_i, \sigma_i^2)}{\sum_l \pi_l \mathcal{N}(x_j | \mu_l, \sigma_l^2)}$$

- Calculate the probability that the observation is originated from a specific component
- For a random $j \in \{1, \dots, n\}$, sample z_j based on the current estimates of mixture parameters.

Initialization

- Must start with an initial assignment of component labels for each observed data point
- A simple choice is to start with random assignment with equal probabilities

The Gibbs Sampler

- Select initial parameters
- Repeat a large number of times
 - Select an element
 - Update conditional on other elements

Implementing Gaussian Mixture Gibbs Sampler

```
class normMixGibbs {
public:
    int k;                // # of components
    int n;                // # of data
    std::vector<double> data; // size n : observed data
    std::vector<double> labels; // size n : label assignment for each observations
    std::vector<double> pis; // size k : pis
    std::vector<double> means; // size k : means
    std::vector<double> sigmas; // size k : sds
    std::vector<int> counts; // size k : # of elements in each labels
    std::vector<double> sums; // size k : sum across each label
    std::vector<double> sumsq; // size k : squared sum across each label
    normMixGibbs(std::vector<double>& _data, int _k); // constructor
    void initParams(); // initialize parameters
    void updateParams(int numObs); // update parameters
    void remove(int i); // remove elements
    void add(int i, int label); // add an element with new label
    int sampleLabel(double x); // sample the label of an element
    void runGibbs(); // run Gibbs sampler
};
```

Update procedure in Gibbs sampler

$$\Pr(z_j = i | x_j, \pi, \lambda) = \frac{\pi_i \mathcal{N}(x_j | \mu_i, \sigma_i^2)}{\sum_l \pi_l \mathcal{N}(x_j | \mu_l, \sigma_l^2)}$$

- Calculate the probability that the observation is originated from a specific component
- For a random $j \in \{1, \dots, n\}$, sample z_j based on the current estimates of mixture parameters.

Sampling label of selected value

```
// x is the data needs a new label assignment
int normMixGibbs::sampleLabel(double x) {
    double p = randu(0,1); // generate a random probability
    // evaluate the likelihood of the observations given parameters
    double lk = NormMix615::dmix(x, pis, means, sigmas);
    // use randomized values to randomly assign labels
    // based on the likelihood contributed by each component,
    for(int i=0; i < k-1; ++i) {
        double pl = pis[i] * NormMix615::dnorm(x, means[i], sigmas[i])/lk;
        if ( p < pl ) return i;
        p -= pl;
    }
    // evaluate only k-1 components and assign to the last one if not found
    return k-1;
}
```

Calculating the parameters at each update

- For each $i \in \{1, \dots, k\}$
 - $n_i = \sum_{j=1}^n I(z_j = i)$
 - $\pi_i = n_i/n$
 - $\mu_i = \sum_{j=1}^n x_j I(z_j = i) / n_i$
 - $\sigma_i^2 = \sum_{j=1}^n I(z_j = i) (x_j - \mu_i)^2 / n_i$
- This procedure takes $O(n)$, which is quite expensive to run over a large number of iterations.
- Is it possible to reduce the time complexity to constant ($O(k)$)?

Constant time update of parameters

```
// update parameters for each components
// having counts, sums, sumsq allows to reduce time complexity
void normMixGibbs::updateParams(int numObs) {
    for(int i=0; i < k; ++i) { // This takes O(k) complexity
        pis[i] = (double)counts[i]/(double)(numObs);
        means[i] = sums[i]/counts[i];
        sigmas[i] = sqrt((sumsq[i]/counts[i] - means[i]*means[i])+1e-7);
    }
}
```

Removing and adding elements

```
// We want to remove object to calculate the conditional distribution
// excluding one component's label information
void normMixGibbs::remove(int i) {
    int l = labels[i];
    --counts[l];
    sums[l] -= data[i];
    sumsqs[l] -= (data[i]*data[i]);
}
// Adding the removed object with newer label assignment
void normMixGibbs::add(int i, int label) {
    labels[i] = label;
    ++counts[label];
    sums[label] += data[i];
    sumsqs[label] += (data[i]*data[i]);
}
```

The Gibbs Sampler

```
void normMixGibbs::runGibbs() {
    initParams(); // initialize label assignments
    for(int i=0; i < 10000000; ++i) { // repeat a large number of times
        int id = randn(0,n); // select a label to update
        if ( counts[labels[id]] < MIN_COUNTS ) continue; // avoid boundary conditions
        remove(id); // remove the elements
        updateParams(n-1); // update pis, means, sigmas
        int label = sampleLabel(data[id]); // sample new label conditionally
        add(id, label); // add the element back with new label

        if ( ( i > BURN_IN ) && ( i % THIN_INTERVAL == 0 ) ) {
            // report intermediate results if needed
            // calculate summary statistics of the parameters to estimate
        }
    }
}
```

Initialization

```
void normMixGibbs::initParams() {  
    // set everything to zero  
    for(int i=0; i < k; ++i) {  
        counts[i] = 0;  
        sums[i] = sumsq[i] = 0;  
    }  
  
    int r;  
    for(int i=0; i < n; ++i) {  
        r = randn(0, k);  
        labels[i] = r;    // assign random labels at the beginning  
        ++counts[r];    // update counts, sums, sumsq accordingly  
        sums[r] += data[i];  
        sumsq[r] += (data[i]*data[i]);  
    }  
}
```

A running example

```
user@host:~/> ./mixSimplex ./mix.dat  
Minimim = 3043.46, at pi = 0.667271,  
between N(-0.0304604,1.00326) and N(5.01226,0.956009)
```

```
user@host:~/> ./mixEM ./mix.dat  
Minimim = 3043.46, at pi = 0.667842,  
between N(-0.0299457,1.00791) and N(5.0128,0.913825)
```

```
user@host:~/> ./mixSA ./mix.dat  
Minimim = 3043.46, at pi = 0.667793,  
between N(-0.030148,1.00478) and N(5.01245,0.91296)
```

```
user@host:~/> ./mixGibbs ./mix.dat  
Minimum = 3043.46, pi = 0.667772,  
between N(-0.0301499,1.00764) and N(5.01214,0.915481)
```

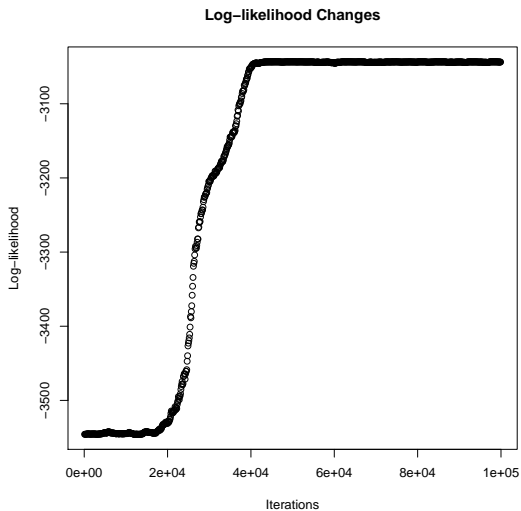
Notes on Gibbs Sampler

- Previous optimizers settled on a minimum eventually
- The Gibbs sampler continues wandering through the stationary distribution
- Forever!

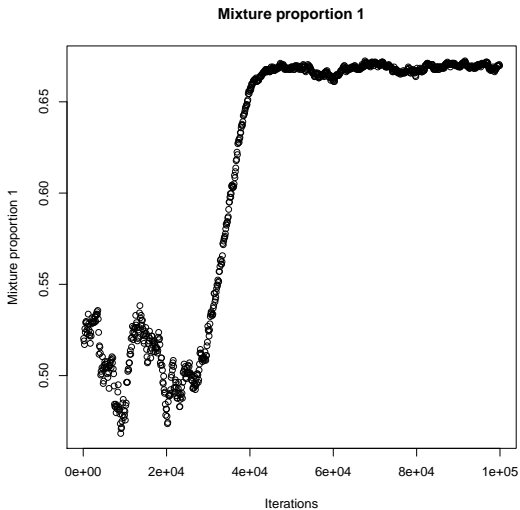
Drawing Inferences

- To draw inferences, summarize parameter values from stationary distribution
- For example, calculate the means or medians.
- Burn-in time required to converge to a reasonable solution before start collecting the estimated parameter values

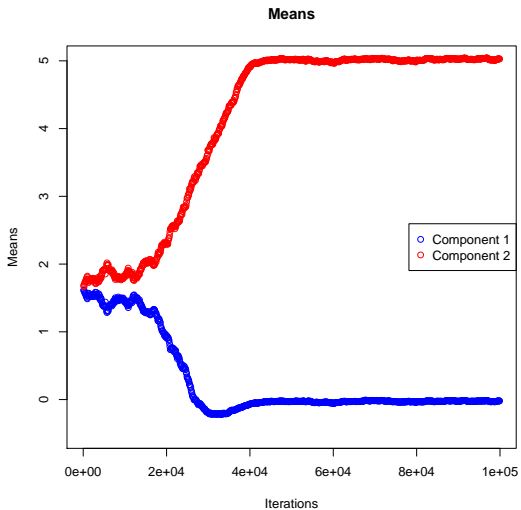
Log-likelihood changes



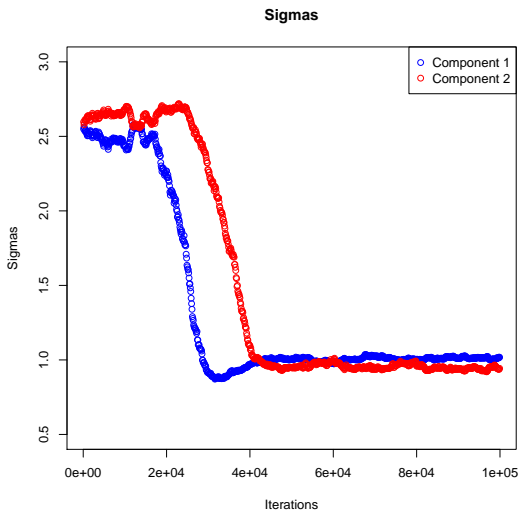
Mixing proportions



Means



Standard deviations



Advantages of Gibbs Samplers

- Gibbs sampler allows to obtain posterior distribution of parameters, conditioned on the observed data.
- Joint distribution between parameters can also be obtained

Connecting Simulated Annealing and Gibbs Sampler

Both Methods are Markov Chains

- The distribution of $\lambda^{(t)}$ only depends on $\lambda^{(t-1)}$
- Update rule defines the transition probabilities between two states, requiring aperiodicity and irreducibility.

Both Methods are Metropolis-Hastings Algorithms

- Acceptance of proposed update is probabilistically determined by relative probabilities between the original and proposed states

Metropolis-Hastings Acceptance Probability

- Let $\theta_{ij} = \Pr(q_t = j | q_{t-1} = i)$
- Let π_i and π_j be the relative probabilities of each state
- The Metropolis-Hasting acceptance probability is

$$a_{ij} = \min \left(1, \frac{\pi_j \theta_{ji}}{\pi_i \theta_{ij}} \right)$$

- We need to know relative ratio between π_j and π_i .
- The equilibrium distribution $\Pr(q_t = i) = \pi_i$ will be reached.

Gibbs Sampler

- The Gibbs sampler ensures that $\pi_i \theta_{ij} = \pi_j \theta_{ji}$
- As a consequence,

$$a_{ij} = \min \left(1, \frac{\pi_j \theta_{ji}}{\pi_i \theta_{ij}} \right) = 1$$

Simulated Annealing

- Given a temperature parameter τ
- π_i are replaced with

$$\pi_i^{(\tau)} = \frac{\pi_i^{1/\tau}}{\sum_j \pi_j^{1/\tau}}$$

- At high temperatures, the probability distribution between the states are flattened.
- At low temperatures, larger weights are given to high probability states

Gibbs Sampler

- MCMC + Metropolis-Hasting Method
- Proposed updates per each parameter based on conditional distribution
- Effective way to obtain joint distribution between the parameters empirically