Biostatistics 615/815 Lecture 19: Expectation-Maximization (EM) Algorithm Simulated Annealing

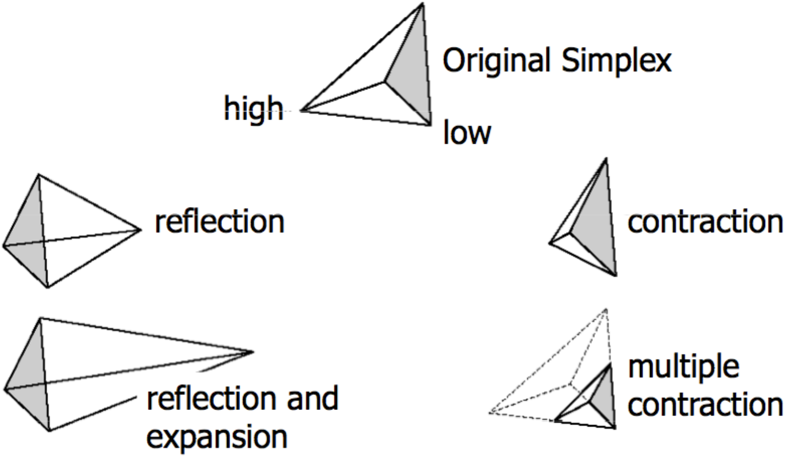
Hyun Min Kang

November 20th, 2012

Recap - The Simplex Method

- General method for optimization
    - Makes few assumptions about function
- Crawls towards minimum using simplex
- Some recommendations
    - Multiple starting points
    - Restart maximization at proposed solution

Summary : The Simplex Method



Implementing Gaussian Mixture : `normMix615.h`

```cpp
class NormMix615 {
 public:
  static double dnorm(double x, double mu, double sigma) {
    return 1.0 / (sigma * sqrt(M_PI * 2.0)) *
           exp (-0.5 * (x - mu) * (x-mu) / sigma / sigma);
  }
  static double dmix(double x, std::vector<double>& pis, std::vector<double>& means,
                     std::vector<double>& sigmas) {
    double density = 0;
    for(int i=0; i < (int)pis.size(); ++i)
      density += pis[i] * dnorm(x,means[i],sigmas[i]);
    return density;
  }
  static double mixLLK(std::vector<double>& xs, std::vector<double>& pis,
                      std::vector<double>& means, std::vector<double>& sigmas) {
    int i=0;
    double llk = 0.0;
    for(int i=0; i < xs.size(); ++i)
      llk += log(dmix(xs[i], pis, means, sigmas));
    return llk;
  }
};
```

Gaussian Mixture Function Object

```
class LLKNormMixFunc {
public:     // below are public functions
  LLKNormMixFunc(int k, std::vector<double>& y) :
      numComponents(k), data(y), numFunctionCalls(0) {}
  // core function - called when foo() is used
  // x is the combined list of MLE parameters (pis, means, sigmas)
  double operator() (std::vector<double>& x);
  std::vector<double> data;
  int numComponents;
  int numFunctionCalls;
};
```

Implementing likelihood of data

```
double LLKNormMixFunc::operator() (std::vector<double>& x) {
// x has (3*k-1) dimensions
  std::vector<double> priors;
  std::vector<double> means;
  std::vector<double> sigmas;
  assignPriors(x, priors); // transform (k-1) real numbers to priors
  for(int i=0; i < numComponents; ++i) {
    means.push_back(x[numComponents-1+i]);
    sigmas.push_back(x[2*numComponents-1+i]);
  }
  return 0-NormMix615::mixLLK(data, priors, means, sigmas);
}
```

Transforming between bounded and unbounded space

```
void LLKNormMixFunc::assignPriors(std::vector<double>& x,
                                  std::vector<double>& priors) {
  priors.clear();
  double p = 1.;
  for(int i=0; i < numComponents-1; ++i) {
    double logit = 1./(1.+exp(0-x[i]));
    priors.push_back(p*logit);
    p = p*(1.-logit);
  }
  priors.push_back(p);
}
```

Probably a better way of transformation

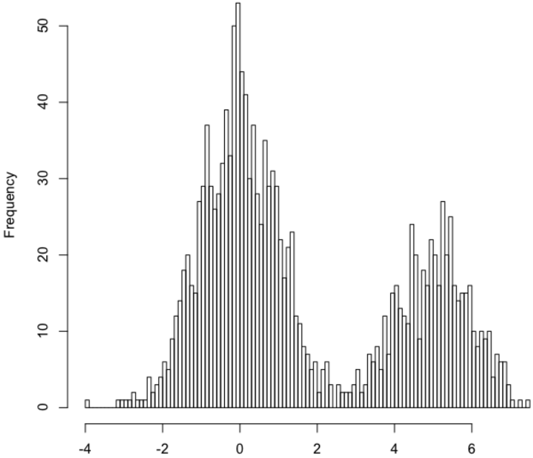
```
void LLKNormMixFunc::assignPriors(std::vector<double>& x,
                                  std::vector<double>& priors) {
  priors.clear();
  double psum = 0, xsum = 0;
  for(int i=0; i < numComponents-1; ++i) {
    double logit = 1./(1.+exp(0-x[i]));
    priors.push_back(logit);
    psum += logit;
    xsum += x[i];
  }
  double pe = 1./(1+exp(xsum));    // probability of last component
  double pec = 1./(1+exp(0-xsum)); // pec = 1-pe

  priors.push_back(pe);
  for(int i=0; i < numComponents-1; ++i)
    priors[i] = priors[i] / psum * pec;
}
```

Simplex Method for Gaussian Mixture

```
#include <iostream>
#include <fstream>
#include "simplex615.h"
#include "normMix615.h"
#include "llkNormMixFunc.h"
#define ZEPS 1e-10
int main(int main, char** argv) {
  double point[5] = {0, -1, 1, 1, 1};   // 50:50 mixture of N(-1,1) and N(1,1)
  simplex615<LLKNormMixFunc> simplex(point, 5);
  std::vector<double> data;                // input data
  std::ifstream file(argv[1]);             // open file
  double tok;                              // temporary variable
  while(file >> tok) data.push_back(tok);  // read data from file
  LLKNormMixFunc foo(2, data);             // 2-dimensional mixture model
  simplex.amoeba(foo, 1e-7);               // run the Simplex Method
  std::cout << "Minimum = " << simplex.ymin() << ", at pi = "
            << (1./(1.+exp(0-simplex.xmin()[0]))) << "," << "between N("
            << simplex.xmin()[1] << "," << simplex.xmin()[3] << ") and N("
            << simplex.xmin()[2] << "," << simplex.xmin()[4] << ")" << std::endl;
  return 0;
}
```

A working example



A working example

Simulation of data

```
> x <- rnorm(1000)
> y <- rnorm(500)+5
> write.table(matrix(c(x,y),1500,1),'mix.dat',row.names=F,col.names=F)
```

A Running Example

```
Minimum = 3043.46, at pi = 0.667271,
between N(-0.0304604,1.00326) and N(5.01226,0.956009)
(305 function evaluations in total)
```

The E-M algorithm

- General algorithm for missing data problem
- Requires ”specialization” to the problem in hand
- Frequently applied to mixture distributions

Some citation records (as of Apr. 2011)

- The E-M algorithm
  - Dempster, Laird, and Rubin (1977) J Royal Statistical Society (B) 39:1-38
  - Cited in over 19,624 research articles
- The Simplex Method
  - Nelder and Mead (1965) Computer Journal 7:308-313
  - Cited in over 10,727 research articles

The Basic E-M Strategy

- $\mathbf{w} = (\mathbf{x}, \mathbf{z})$
  - Complete data **w** - what we would like to have
  - Observed data **x** - individual observations
  - Missing data **z** - hidden / missing variables
- The algorithm
  - Use estimated parameters to infer **z**
  - Update estimated parameters using **x**
  - Repeat until convergence

The E-M Strategy in Gaussian Mixtures

When are the E-M algorithms useful?

- Problem is simpler to solve for complete data
  - Maximum likelihood estimates can be calculated using standard methods
- Estimates of mixture parameters would be obtained straightforwardly
  - if the origin of each observation is known

Filling in Missing Data in Gaussian Mixtures

- Missing data is the group assignment of each observation
- Complete data generated by assigning observations to groups 'probabilistically'

E-M formulation of Gaussian Mixture

- Gaussian mixture distribution given $\theta = (\pi, \mu, \sigma)$.

$$p(x_i) = \sum_{k=1}^{K} \pi_K \mathcal{N}(x_i | \mu_k, \sigma_k^2)$$

- Introducing latent variable $\mathbf{z}$
  - $z_i \in \{1, \cdots, K\}$ is class assignment
- The marginal likelihood of observed data

$$L(\theta; \mathbf{x}) = p(\mathbf{x}|\theta) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}|\theta)$$

  is often intractable
- Use complete data likelihood to approximate $L(\theta; \mathbf{x})$

The E-M algorithm

Expectation step (E-step)

- Given the current estimates of parameters $\theta^{(t)}$, 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

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

Maximization step (M-step)

- Find the parameter that maximize the expected log-likelihood

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

Implementing Gaussian Mixture E-M

```
class normMixEM {
public:
  int k;                        // # of components
  int n;                        // # of data
  std::vector<double> data;     // observed data
  std::vector<double> pis;      // pis
  std::vector<double> means;    // means
  std::vector<double> sigmas;   // sds
  std::vector<double> probs;    // (n*k) class probability
  normMixEM(std::vector<double>& input, int _k);
  void initParams();
  void updateProbs();           // E-step
  void updatePis();             // M-step (1)
  void updateMeans();           // M-step (2)
  void updateSigmas();          // M-step (3)
  double runEM(double eps);
};
```

Gaussian mixture : The E-step

Key idea

- Estimate the missing data - 'class assignment'
- By conditioning on current parameter values
- Basically, "classify" each observation to the best of current step.

Gaussian mixture : The E-step

Key idea

- Estimate the missing data - 'class assignment'
- By conditioning on current parameter values
- Basically, "classify" each observation to the best of current step.

Classification Probabilities

$$\Pr(z_i = j | x_i, \pi, \mu, \sigma) = \frac{\pi_j \mathcal{N}(x_i | \mu_j, \sigma_j^2)}{\sum_k \pi_k \mathcal{N}(x_i | \mu_k, \sigma_k^2)}$$

Implementation of E-step

```
void normMixEM::updateProbs() {
  for(int i=0; i < n; ++i) {
    double cum = 0;
    for(int j=0; j < k; ++j) {
      probs[i*k+j] = pis[j]*NormMix615::dnorm(data[i],means[j],sigmas[j]);
      cum += probs[i*k+j];
    }
    for(int j=0; j < k; ++j) {
      probs[i*k+j] /= cum;
    }
  }
}
```

Mixture of Normals : The M-step

- Update mixture parameters to maximize the likelihood of the data
- Becomes simple when we assume that the current class assignment are correct
- Simply use the same proportions, weighted means and variances to update parameters
- This step is guaranteed never to decrease the likelihood

Updating Mixture Proportions

$$\pi_k = \frac{\sum_i^n \Pr(z_i = k | x_i, \mu, \sigma^2)}{n}$$

- Count the observations assigned to each group

Updating Mixture Proportions - Implementations

```
void normMixEM::updatePis() {
  for(int j=0; j < k; ++j) {
    pis[j] = 0;
    for(int i=0; i < n; ++i) {
      pis[j] += probs[i*k+j];
    }
    pis[j] /= n;
  }
}
```

Updating Component Means

$$
\begin{aligned}
\hat{\mu}_k &= \frac{\sum_i x_i \Pr(z_i = k | x_i, \mu, \sigma^2)}{\sum_i \Pr(z_i = k | x_i, \mu, \sigma^2)} \\
&= \frac{\sum_i x_i \Pr(z_i = k | x_i, \mu, \sigma^2)}{n\pi_k}
\end{aligned}
$$

- Calculate weighted mean for group
- Weights are probabilities of group membership

Updating Component Means - Implementations

```
void normMixEM::updateMeans() {
  for(int j=0; j < k; ++j) {
    means[j] = 0;
    for(int i=0; i < n; ++i) {
      means[j] += data[i] * probs[i*k+j];
    }
    means[j] /= (n * pis[j] + TINY);
  }
}
```

Updating Component Variances

$$\sigma_k^2 = \frac{\sum_i (x_i - \mu_k)^2 \Pr(z_i = k | x_i, \mu, \sigma)}{n\pi_k}$$

- Calculate weighted sum of squared differences
- Weights are probabilities of group membership

Updating Component Variances - Implementations

```
void normMixEM::updateSigmas() {
  for(int j=0; j < k; ++j) {
    sigmas[j] = 0;
    for(int i=0; i < n; ++i) {
      sigmas[j] += (data[i]-means[j])*(data[i]-means[j])*probs[i*k+j];
    }
    sigmas[j] = sqrt(sigmas[j] / (n * pis[j] + TINY)); // TINY can be small, e.g.
  }
}
```

E-M Algorithm for Mixtures

1. Guesstimate starting parameters
2. Use Bayes' theorem to calculate group assignment probabilities
3. Update parameters using estimated assignments
4. Repeat steps 2 and 3 until likelihood is stable

Implementation of E-M algorithm - putting things together

```
double normMixEM::runEM(double eps) {
  double llk = 0, prevLLK = 0;
  initParams();
  while( ( llk == 0 ) || ( check_tol(llk, prevLLK, eps) == 0 ) ) {
    updateProbs();
    updatePis();
    updateMeans();
    updateSigmas();
    prevLLK = llk;
    llk = NormMix615::mixLLK(data, pis, means, sigmas);
  }
  return llk;
}
```

Constructing `normMixEM` object

```
normMixEM::normMixEM(std::vector<double>& input, int _k) {
  data = input;
  k = _k;
  n = (int)data.size();
  pis.resize(k);
  means.resize(k);
  sigmas.resize(k);
  probs.resize(k * data.size());
}
```

Initializing the parameters

```
void normMixEM::initParams() {
  double sum = 0, sqsum = 0;
  for(int i=0; i < n; ++i) {
    sum += data[i];
    sqsum += (data[i]*data[i]);
  }
  double mean = sum/n;
  double sigma = sqrt(sqsum/n - sum*sum/n/n);
  for(int i=0; i < k; ++i) {
    pis[i] = 1./k;               // uniform priors
    means[i] = data[rand() % n]; // pick random data points
    sigmas[i] = sigma;           // pick uniform variance
  }
}
```

A working example

main() function

```
int main(int main, char** argv) {
  std::vector<double> data;
  std::ifstream file(argv[1]);
  double tok;
  while(file >> tok) data.push_back(tok);
  normMixEM em(data,2);
  double minLLK = em.runEM(1e-6);
  std::cout << "Minimum = " << minLLK << ", at pi = " << em.pis[0] << ","
            << "between N(" << em.means[0] << "," << em.sigmas[0]<< "^2) and N("
            << em.means[1] << "," << em.sigmas[1] << "^2)" << std::endl;
  return 0;
}
```

Running example

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

Summary : The E-M Algorithm

- Iterative procedure to find maximum likelihood estimate
    - E-step : Calculate the distribution of latent variables and the expected log-likelihood of the parameters given current set of parameters
    - M-step : Update the parameters based on the expected log-likelihood function
- The iteration does not decrease the marginal likelihood function
- But no guarantee that it will converge to the MLE
- Particularly useful when the likelihood is an exponential family
    - The E-step becomes the sum of expectations of sufficient statistics
    - The M-step involves maximizing a linear function, where closed form solution can often be found

Summary : The E-M Algorithm

- Iterative procedure to find maximum likelihood estimate
  - E-step : Calculate the distribution of latent variables and the expected log-likelihood of the parameters given current set of parameters
  - M-step : Update the parameters based on the expected log-likelihood function
- The iteration does not decrease the marginal likelihood function
- But no guarantee that it will converge to the MLE
- Particularly useful when the likelihood is an exponential family
  - The E-step becomes the sum of expectations of sufficient statistics
  - The M-step involves maximizing a linear function, where closed form solution can often be found

Local and global optimization methods

Local optimization methods

- "Greedy" optimization methods
    - Can get trapped at local minima
    - Outcome might depend on starting point
- Examples
    - Golden Search
    - Nelder-Mead Simplex Method
    - E-M algorithm

Today

- Simulated Annealing
- Markov-Chain Monte-Carlo Method
- Designed to search for global minimum among many local minima

Local minimization methods

The problem

- Most minimization strategies find the *nearest* local minimum from the starting point
- Standard strategy
    - Generate trial point based on current estimates
    - Evaluate function at proposed location
    - Accept new value if it improves solution

The solution

- We need a strategy to find other minima
- To do so, we sometimes need to select new points that does not improve solution
- How?

Simulated Annealing

Annealing

- One manner in which crystals are formed
- Gradual cooling of liquid
    - At high temperatures, molecules move freely
    - At low temperatures, molecules are "stuck"
- If cooling is slow
    - Low energy, organized crystal lattice formed

Simulated Annealing

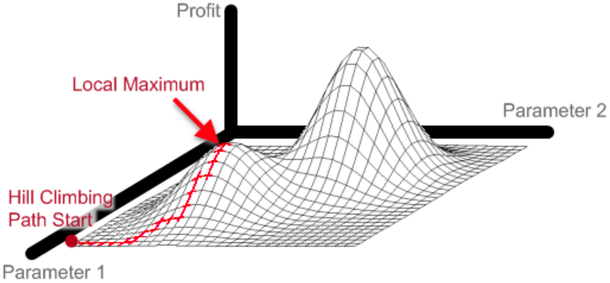
- Analogy with thermodynamics
- Incorporate a temperature parameter into the minimization procedure
- At high temperatures, explore parameter space
- At lower temperatures, restrict exploration

Simulated Annealing Strategy

- Consider decreasing series of temperatures
- For each temperature, iterate these step
    - Propose an update and evaluation function
    - Accept updates that improve solution
    - Accept some updates that don't improve solution
        - Acceptance probability depends on "temperature" parameter
- If cooling is sufficiently slow, the global minimum will be reached

Local minimization methods

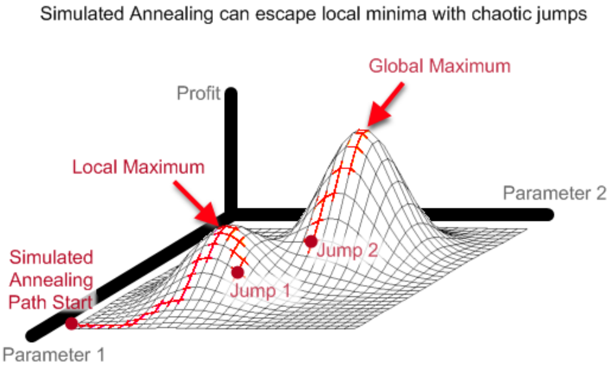
The problem with hill climbing is that it gets stuck on "local-maxima"



Images by Max Dama from

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

Global minimization with Simulated Annealing



Images by Max Dama from

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