Biostatistics 615/815 Homework Highlights

Hyun Min Kang

December 8th, 2011

Homework 2-1: Quicksort Pivoting

```
void quickSort(std::vector<int>& A, int p, int r, int& numcomp) {
  if ( p < r ) {
    // this part corresponds to PARTITION algorithm
    //int pivIdx = l; // ** LINE TO EDIT **  CURRENT PIVOT IS RIGHTMOST ELEMENT
    int m = (p+r)/2;
    int pivIdx = (A[r] > A[p]) ?
                   ( A[p] > A[m] ? p : ( A[r] > A[m] ? m : r ) )
                 : ( A[r] > A[m] ? r : ( A[p] > A[m] ? m : p ) );

    int pivVal = A[pivIdx];
    ...
  }
}
```

Homework 3-1: Dynamic Programming

```
#include <iostream>
int binom(int n, int k, int** stored) {
  if ( stored[n][k] > 0 ) {
    return stored[n][k];
  }
  else if ( ( k == 0 ) || ( k == n ) ) {
    stored[n][k] = 1;
    return stored[n][k];
  }
  stored[n][k] = binom(n-1,k,stored) + binom(n-1,k-1,stored);
  return stored[n][k];
}
```

Homework 4-2

```
// sample from multinomial distribution where the probabilities
// are defined as probs (sum of probs should be 1)
// x is randomly distributed between 0 and 1
int multiSample(double x, std::vector<double>& probs) {
  for(int i=0; i < probs.size(); ++i) {
    x -= probs[i];
    if ( x <= 0 ) return i;
  }
  return probs.size()-1;
}
```

Homework 5-1 : Simplex Algorithm

```
class squaredNorm {
public:
  squaredNorm() : numCalls(0) {}
  virtual double operator() (std::vector<double>& x) {
    double sum = 0;
    for(int i=0; i < (int)x.size(); ++i) {
      sum += x[i]*x[i];
    }
    ++numCalls;
    return sum;
  }
  int numCalls;
};

int main(int argc, char** argv) {
  ...
  simplex615<squaredNorm> simplex(p, n);
  squaredNorm foo;
  simplex.amoeba(foo, 1e-7);
  std::cout << simplex.ymin() << "\t" << foo.numCalls << std::endl;
  return 0;
}
```

Homework 5-3

```
$ ./normMixAlphaEstimate
Usage : ./normMixAlphaEstimate [infile] [mean1] [sd1] [mean2][sd2]

$ ./normMixSimul 10000 0.8 0 1 5 1 > input.txt

$./normMixAlphaEstimate input.txt 0 1 5 1
Golden Search  : alpha = 0.794469, LLK = -19139, niter = 31
Brent Algorithm: alpha = 0.794471, LLK = -19139, niter = 11
E-M Algorithm  : alpha = 0.794466, LLK = -19139, niter = 3
```

hw5-3.cpp

```
int main(int argc, char** argv)
{
  std::vector<double> obs;
  readFromFile( obs, argv[1] );
  double mean1 = atof(argv[2]);   double mean2 = atof(argv[3]);
  double sigma1 = atof(argv[4]);  double sigma2 = atof(argv[5]);

  llkNormMixAlpha llk(obs, mean1, mean2, sigma1, sigma2); // LLK function

  goldenSearch (llk,0,1,0.5,1e-6); // goldenSearch, print inside

  typedef std::pair<double, double> Result; // Brent
  boost::uintmax_t niter;
  Result r2 = boost::math::tools::brent_find_minima(llk, 0.0, 1.0, 20, niter);
  std::cout << "Brent : x=" << r2.first << ", f=" << r2.second
            << ", niter=" << niter << std::endl;

  normMixEMAlpha em( obs, mean1, mean2, sigma1,sigma2); // E-M
  double emLLK = em.runEM(1e-7);
  std::cout << "E-M : x=" << em.pis[0] << ", f=" << emLLK
            << ", niter=" << em.mixLLK.numFunctionCalls << std::endl;
  return 0;
}
```

llkNormMixAlpha.h

```
#include "normMix615.h"
class llkNormMixAlpha {
public:
  std::vector<double> data;
  double mean1, mean2, sigma1, sigma2;
  int numFunctionCalls;

  llkNormMixAlpha(std::vector<double>& y, double m1, double m2, double s1, double s2)
     : data(y), mean1(m1), mean2(m2), sigma1(s1), sigma2(s2),
       numFunctionCalls(0) {}

  double operator() (double alpha) {
    std::vector<double> priors, means, sigmas;
    priors.resize(2); priors[0] = alpha; priors[1] = 1.-alpha;
    means.resize(2); means[0] = mean1; means[1] = mean2;
    sigmas.resize(2); sigmas[0] = sigma1; sigmas[1] = sigma2;
    double llk = 0-normMix615::mixLLK(data, priors, means, sigmas);
    ++numFunctionCalls;
    return llk;
  }
};
```

normMixEMAlpha.h

```
class normMixEMAlpha {
public:
  llkNormMixAlpha mixLLK;
  std::vector<double> probs;
  std::vector<double> pis;    // pis
  normMixEMAlpha(std::vector<double>& input, double m1, double m2, double s1, double s2)
     : mixLLK(input, m1, m2, s1, s2) {
    pis.resize(2);
    probs.resize(2 * input.size());
    pis[0] = pis[1] = .5;
  }
  void updateProbs();         // E-step
  void updatePis();           // M-step 1
  int numCalls;
  double runEM(double eps);
};
```

normMixEMAlpha.h

```
double normMixEMAlpha::runEM(double eps) {
  double llk = 0, prevLLK = -1e9;
  numCalls = 0;
  while( check_tol(llk, prevLLK, eps) == 0 ) {
    updateProbs();
    updatePis();
    prevLLK = llk;
    llk = mixLLK(pis[0]);
  }
  return llk;
}
```

normMixEMAlpha.h

```
void normMixEMAlpha::updateProbs() {
  int n = (int)mixLLK.data.size();
  for(int i=0; i < n; ++i) {
    double cum = 0;
    probs[i*2+0] = pis[0] *
         normMix615::dnorm(mixLLK.data[i],mixLLK.mean1,mixLLK.sigma1);
    probs[i*2+1] = pis[1] *
         normMix615::dnorm(mixLLK.data[i],mixLLK.mean2,mixLLK.sigma2);
    cum = probs[i*2] + probs[i*2+1];
    for(int j=0; j < 2; ++j) {
      probs[i*2+j] /= cum;
    }
  }
}
```

normMixEMAlpha.h

```
void normMixEMAlpha::updatePis() {
  int n = (int)mixLLK.data.size();
  int k = 2;
  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;
  }
}
```