ANLY-601 Spring 2018

Assignment 6
Due April 24 in class

1. **Fitting Gaussian Mixture Models**

   This is an exercise in fitting mixtures of Gaussians to data. I will have you fit standard mixture models and mixture models with *spherically symmetric* covariance matrices (as you explored in the last homework for single Gaussians. You will need to write code for the EM algorithm. *I encourage you to write, test, and debug the code in groups.* But please run your own experiments and produce your own reports.

   On the Blackboard page is a dataset *toydata1.txt* consisting of 1500, 2-d vectors. One vector per row.

   (a) Make a scatter plot of the data and fit models with 2,3,4,5, ... up to 10–20 Gaussian components. Fit the model on the first 500 data vectors using the EM procedure.

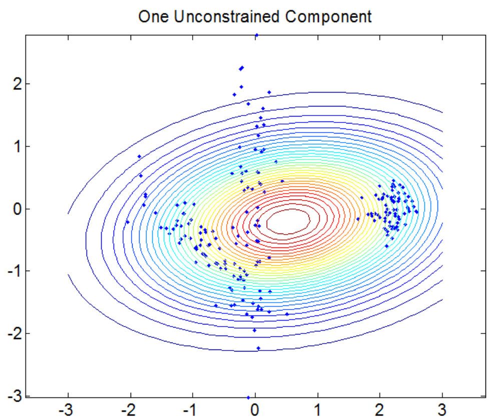
   Make plots of the log-likelihood per data point

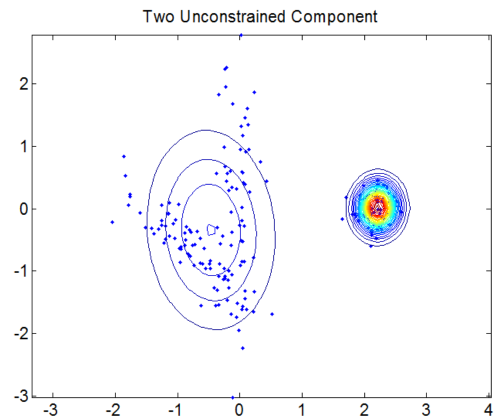
   $$\mathcal{L} = \frac{1}{N}\sum_{a=1}^{N} \ln p(x^{(a)}|\Theta)$$

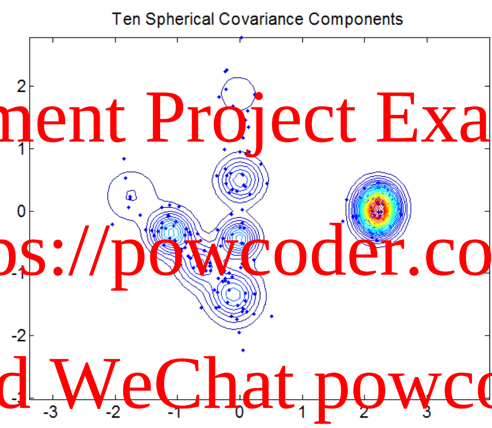
   on the training set vs the number of mixture components. (Here Θ refers to the collection of all the model parameters α_k, μ_k, Σ_k, $k = 1 \ldots L$, with L the number of mixture components.) Since the EM algorithm finds *local* optima, it's useful to try several random initializations for each model. (For each model, keep the parameters Θ from the run with the *highest* $\mathcal{L}$ on the training set, as that's the best fit.)

   Also make plots of the log-likelihood per data point on the holdout set (the remaining 1000 points in the data set) for each model trained. Contrast how the log-likelihood on the training set behaves as the number of components is increased with how the log-likelihood on the holdout set behaves. For several of your solutions, compare a scatter plot of the data with either a contour plot, or a surface plot of the model density. (I've included three examples below.) Comment on the results.

   (b) Repeat the exercise for a model built with *spherical* Gaussian mixture components — i.e. components with covariance matrices $\sigma_k^2 I$, $k = 1 \ldots L$. As before, make plots of the log-likelihood per data point on the training set vs the number of mixture components. Repeat for the log-likelihood per data point on the holdout set. Contrast how the log-likelihood on the training set behaves as the number of components is increased with how the log-likelihood on the holdout set behaves. And contrast the results from this part with those for the unconstrained model in part (a). For several of your solutions, compare a scatter plot of the data with either a contour plot, or a surface plot of the model density. Comment on the results.


One Unconstrained Component


Two Unconstrained Component


Ten Spherical Covariance Components

Here's pseudocode (nearly full code!) for the EM procedure to fit the mixture of Gaussian models. It follows the procedure given in the lecture notes (but includes a regularizer)— so re-read those too.

```
Load training data D = { x_a, a=1...N }    % N samples, each of dimension dim
Set Number of Components L
Intialize:
   for i=1...L
    alpha_i = 1/L
    mu_i = choose randomly from data points,
    Sigma_i = IdentityMatrix
   end; i

logLike = log p(D|{alpha_i, mu_i, Sigma_i, i=1...L} )
oldLogLike = logLike-1

While ( | oldLogLike - logLike | > 0.00001 ) AND ( iterations < 500 )

  % E-Step
   for a=1 ... N
    p( x_a ) = sum( alpha_j*p(x_a | mu_j, Sigma_j), j=1...L )
    for i=1...L
      h_ia = alpha_i * p( x_a | mu_i, Sigma_i )/ p(x_a)
      % This regularization step prevents singularities
      epsilon = 1E-8
      h_ia = ( h_ia + epsilon ) / ( 1 + L*epsilon )
    end % for i
   end % for a

  % M-Step
  for i = 1 ... L
   alpha_i = 1/N *sum ( h_ia, a=1...N )
   pointsInI = sum( h_ia, a=1...N )
   mu_i = sum( h_ia * x_a, a=1...N ) / pointsInI
   if UnConstrained Covariance
    Sigma_i = sum( h_ia * ( x_a - m_i )*Transpose(x_a - m_i), a=1...N ) / pointsInI
    % Note (x_a - m_i) is a COLUMN vector
   else % Sigma_i = sigma2_i*IdentityMatrix
     sigma2_i = sum ( h_ia * Transpose(x_a-m_i)*(x_a-m_i), a=1...N ) / (pointsInI*dim)
      % Note this is 1/dim*Trace(SigmaUnconstrained) and you can compute it this way
     Sigma_i = sigma2_i * IndentityMatrix
  end % for i

oldLogLike = logLike
logLike = log p(D| { alpha_i, mu_i, Sigma_i, i=1...L } )

end % While
```