

# CS 274A, Homework 6: EM and Gaussian Mixtures

Winter 2019

Due Date: 6pm Friday March 15th 2019

Submit code to EEE Dropbox and report in PDF to Gradescope

## General Instructions

- Required reading is the Note Set on Mixture Models and the EM Algorithm on the class Web page. Optional additional reading is Section 20.3 on Gaussian mixtures in the Barber text and pages 351–357 in the Murphy text.
- This homework includes quite a bit of programming. If you take it step by step it is straightforward, but be sure to start working on this well in advance of the deadline.
- In addition to the data, the homework directory includes two subdirectories with sample code and templates, one for Python and one for Matlab, to help you get started.

## MATLAB: K-Means and Gaussian Mixture Learning

### Data Sets

There are 3 data sets for experimentation: all are two-dimensional so that you can visualize the results. The data format is one data vector per row, each column corresponds to a feature. The data sets are available from the class Web page. Data set 3 also has class labels, called `labelset3`, which are not to be used in clustering but can be used for exploring your results after you do clustering. You should read the data into your Python/R/Matlab/etc environment and plot some scatter plots, before you do any clustering, just so that you have an idea of what the data sets look like in 2-d. The directory for the homework on the class Web page contains some Python and Matlab scripts for plotting the data and model parameters to help get you started.

1. Data set 1: this is 2d data simulated from 2 Gaussians, with a fair degree of overlap. The first 800 points belong to 1 Gaussian, the 2nd to the other Gaussian.
2. Data set 2: this is 2d data simulated from 3 Gaussians, with considerable overlap. There are 500 points from each Gaussian, ordered together in the file.

3. Data set 3: this is a real 2d data set from a past collaboration with Professor Christine McLaren, Department of Epidemiology, UCI. Each data point represents an individual person: one group is normal healthy people, the other has iron deficient anemia (so the true value of  $K$  is 2). The two measurements on each individual are their mean red blood-cell volume and their mean hemoglobin concentration. Both measurements tend to be lower for anemic individuals. The class labels are provided in the file `labelset3`.

For the data sets above you are to run the clustering algorithms in an unsupervised manner, i.e., without class labels or knowledge of what the true classes are. You can then use the labels after clustering if you wish to check to see how the clustering matches with “truth.”

4. Image Data: (Optional, this is just for fun if you wish to play with this data—no need to submit anything related to this data set). This data is contained in the subdirectory for “clown image data” and consists of the pixels in a small JPG image `clown.jpg`. You can convert this to an array where each row corresponds to a pixel and there are 3 columns, one for the r, g, and b channels of color information. The image is small to keep the number of pixels to a manageable size—but there are still over 30,000 pixels in total, so unless your code is fairly efficient it is likely to run slowly on this data. There are no “true classes” here for the pixels. K-means and/or Gaussian mixtures can be used to compress the real-values in (r,g,b) into a “dictionary” of  $K$  clusters (effectively using clustering to perform data compression). You could experiment with different values of  $K$  and look at the visual difference between the compressed image (where you replace each pixel by the mean of the cluster it was assigned to) and the original image, as a function of  $K$ . The directory with the data contains a simple Matlab function to read in and to display the (r,g,b) pixel values (if you are using Matlab).

You should randomize the order of the rows for all of these data sets before you run your clustering algorithms, to avoid any possible bias from your algorithm (e.g, during initialization) due to systematic effects based on row order.

## Algorithm 1: $K$ -means Clustering

Implement the standard version of the algorithm as described in the Note Set on Mixture Models and EM. The initial starting points for the  $K$  cluster means can be  $K$  randomly selected data points. You should have an option to run the algorithm  $r$  times from  $r$  different randomly chosen initializations (e.g.,  $r = 10$ ), where you then select the solution that gives the lowest sum of squares error over the  $r$  runs.

## Algorithm 2: Gaussian Mixture Clustering

Gaussian mixture clustering is a probabilistic approach to clustering where we assume a Gaussian mixture model of  $K$  components for the data and we find the parameters of the model using the EM algorithm as described in the Note Set on Mixture Models and EM.

You should use the following general outline for your Gaussian mixture code:

- Initialize the algorithm using one of the methods described in the notes.
- Execute E and M steps as long as the convergence condition is not satisfied:
  - E-step: compute membership probabilities using the current  $\theta^{current}$  values.
  - M-step: compute new parameters  $\theta^{new}$  using the membership probabilities from the E-step
- After each EM iteration compute the log-likelihood of the data using  $\theta^{new}$  (see Notes for a definition of the log-likelihood). This will allow you to print out the log-likelihood values from each EM iteration, as the algorithm is running, to monitor its convergence.
- Check for convergence using one of the methods described in the class notes on the EM algorithm (on the class Web site). If the convergence criterion is not satisfied, then execute another EM iteration.

I also recommend that your algorithm runs this process multiple times from  $r$  different randomly-chosen starting conditions (e.g.  $r = 10$ ) and pick the solution that results in the highest log-likelihood (since EM in general only finds local maxima).

The EM algorithm for Gaussian mixtures is a non-trivial algorithm to get working properly: please try and debug it carefully. Check that the likelihood is non-decreasing at each step (i.e., have your code print it out as it goes through each iteration—if the log-likelihood ever decreases you have a bug in your code). You could also write out the final parameters for the components and check that the estimated parameters are roughly equal to the true parameters (obtained using the labels).

For these data sets you could also impose a maximum number of iterations to halt the algorithm (e.g., 500) if it gets that far and still has not converged. This is particularly handy when debugging!

**Avoiding singular solutions:** the likelihood can go to  $\infty$  if the determinant of the covariance matrix for any component goes to zero (e.g., if any individual  $\sigma_{ii} \rightarrow 0$ ). This typically doesn't happen for small values of  $K$  on these data sets but can start to happen for larger  $K$ . You will need to implement some scheme to prevent such singular (and useless) solutions, especially for the BIC part of the assignment where you will be fitting larger  $K$  values. One somewhat ad hoc workaround (that works well in practice) is to constrain all covariance diagonal terms during the M-step to be greater than some small threshold  $\epsilon$  (e.g.,  $10^{-4}$  times the variance for that dimension, as calculated on the whole data). A simple way to do this is to calculate the  $\Sigma$ 's in the standard M-step manner and then to check each diagonal entry: if any are less than the threshold, then replace them with the threshold.

## General advice on software implementations

In coding your algorithms you should try to make your code as modular as possible, by defining relatively short general functions that can be used in different places in the code. For example, you can define a simple function that calculates the Euclidean distance between a vector  $x$  of dimension  $1 \times d$  and each row

of a matrix of dimension  $n \times d$ . This is one the basic steps that is frequently executed in the  $K$ -means algorithm. Similarly you can define specific functions for initialization, the E-step, the M-step, etc., for Gaussian mixtures below, where each of these functions may call more basic functions, such as a function that evaluates the Gaussian density values for all rows of a data matrix given a set of Gaussian parameters. By making your code modular in this fashion you can test individual functions independently from the rest of the code, your code will be much easier to read and debug, and you will be able to re-use the same functions in different places.

You should also write your equations in using vector and matrix notation where possible, and avoid for-loops when you can. This will usually make your code simpler and faster (especially for languages like Matlab).

For exchanging information about parameters between functions (e.g., sending the parameters learned by the EM learning algorithm to a different function to plot them) you may want to use a data structure that contains all the parameter information, e.g., in MATLAB you can use a structure array with fields such as `gparams(1).mean`, `gparams(1).covariance`, etc.

The Python and MATLAB directories for this homework each contain a simple template for Python and MATLAB code to run Gaussian mixtures—you do not need to necessarily use this, it is just provided here to give you an idea of how you might set up your code.

Finally, you may want to simulate a simple dataset with two well-separated Gaussian components (in 1d or 2d), maybe with 50 or 100 points in each, and use this to test your code, before moving on to the datasets in the homework. If the Gaussians are well-separated (e.g., difference between the two means is roughly 3 sigma apart say) then the EM algorithm should converge quickly—and you can check that the solutions match the true solutions (parameter values) that were used to simulate data.

### Algorithm 3: Automatically Selecting $K$ for Gaussian Mixture Clustering

There are a number of different methods for trying to find  $K$  automatically from the data. A very simple approach (but useful) for doing this is the BIC criterion, i.e, choose  $K$  such that

$$l(D|\hat{\theta}) - \frac{p_K}{2} \log n$$

is maximized, where  $l(D|\hat{\theta})$  is the maximizing value of the log-likelihood as found by EM using data  $D$  with  $K$  components (this value of the log-likelihood can be returned by your Gaussian mixture code),  $p_K$  is the total number of parameters in mixture model  $K$  (you will need to compute this for each  $K$ ), and  $n$  is the number of data points in  $D$ . There is a rather general theoretical justification for this BIC criterion which loosely speaking says that as  $K$  increases we should penalize the log-likelihood of the model with  $K$  components according to its increased complexity, and  $\frac{p_K}{2} \log n$  can be shown to be a good approximation to a true Bayesian penalty term. For mixture models the theory does not hold in general, but nonetheless it can be a useful approximate technique for model selection with mixtures.

To write a program to do this is very simple given your EM Algorithm that you have already written. Simply have a loop that runs the EM algorithm with values of  $K$  going from 1 to some  $K_{\max}$  where  $K_{\max}$  is selected by the user. Note that  $K = 1$  is important: it might be the case that the data are best explained by a single Gaussian! (for the case of  $K = 1$  you obviously don't need to run EM). Your code should return a list, for each value of  $K$ , of both the log-likelihood (which should increase monotonically as  $K$  increases), the BIC criterion as defined above, and the  $K$  value that maximizes the BIC criterion. For some of the smaller data sets (e.g., data set 3) you may find that as  $K$  increases you have more problems both with local maxima of the likelihood function and with singular solutions (so you may need to either make  $K_{\max}$  smaller, and/or run a larger number of random restarts).

There are other options besides BIC for selecting  $K$  for mixture models, if you are interested in this problem of model selection (purely for your own interest, will not be graded, but feel free to submit results of experiments with these methods if you wish—only do this if you have time and if you are sure you have completed the rest of the assignment):

1. A data-driven approach us to use cross-validation on the log-likelihood scores for different  $K$  values (as described in ‘Model selection for probabilistic clustering using cross-validated likelihood’, P. Smyth, *Statistics and Computing*, 2000).
2. A fully Bayesian approach to selecting  $K$  is to compute the marginal likelihood for different values of  $K$  by averaging over parameter values. The marginal likelihood can be approximated by sampling-based techniques such as Markov Chain Monte Carlo (MCMC) methods. Computing marginal likelihoods can be quite tricky for mixture models.
3. Another option is to use *nonparametric Bayesian methods*, which essentially assume that the number of mixture components could be infinitely large and then infer likely values for  $K$  conditioned on the observed data. Kevin Murphy's text for example describes a collapsed Gibbs sampling method for Gaussian mixtures using this approach, in Algorithm 25.1 in his textbook.

## What to Submit

### 1. Code:

- Submit your code for all the functions you have written to the Homework 6 dropbox in EEE.
- Also include a script called `assignment6.py` (for Python), or something similarly named for another language), that loads a data set and runs your code for fitting a model with some fixed value of  $K$  (e.g.,  $K = 2$ ). This script should also run your BIC code to find the best value for  $K$  for the data set. If you are using something other than Python or R or Matlab please also provide a README file to explain to us how to use your code.

### 2. Written Solutions: Submit a PDF copy of your answers and graphs for the questions below. **For each data set**, using the true $K$ value for that data set, show the following

- (a) The  $K$ -means solution (scatter plot in two dimensions (any two dimensions for Data Set 4) illustrating the location of the solution (i.e., the cluster means), and plotting the data from different clusters with different symbols (and/or in color if you would like to use color).
- (b) A plot of the sum-squared-error (divided by  $n$ ) as a function of iteration number in the  $K$ -means algorithm.
- (c) The initial parameter values and the final parameter values (2 plots, each showing means and covariances for each cluster) for the EM/Gaussian mixtures code for the highest-likelihood solution.
- (d) A plot of the log-likelihood (for one run of your algorithm) as a function of iteration number during EM (please use natural logs, i.e., log base  $e$ , in all your calculations and plots so that everyone's results are on the same scale).
- (e) Add some brief comments (1 paragraph) on the difference between  $K$ -means and EM for each data set.
- (f) Generate a table of log-likelihood and BIC scores for  $K$  going from  $K = 1$  to some maximum value (e.g.,  $K = 5, 10$ ). Comment briefly on the results.

In your solutions please try to put multiple plots for the same data set and the same algorithm on the same page (where different pages could correspond to different data sets and different algorithms), to make it easier for us to read your report.