EECS 349 (Machine Learning) Homework 6

How to submit your homework

1. Create a **PDF document** containing answers to the homework questions.
2. Include source code for the program you write.
3. Compress all of the files specified into a .zip file.
4. Name the file in the following manner, *firstname_lastname_hw6.zip*.
3. Submit this .zip file via Canvas by the date specified on Canvas.

1) Making Gaussian Mixture Models (8 points)

Load "gmm_test.csv" and "gmm_train.csv". This contains two labeled samples of data: X_test and X_train. The labels for each sample are contained in Y_test and Y_train. So Y_train(i) contains the label for X_train(i). Each of these samples was drawn from data coming from two different underlying populations, labeled as classes "1" and "2".

In this problem, you will model each class in the training data, using a GMM for each class. You will then design a classifier to classify X_test using your GMMs. You will evaluate your classification results on the test data. You need to submit your source code of "gmmest.py" and "gmmclassify.py" as described below.

Note: You are welcome to make your functions work for multivariate (a.k.a. multidimensional) GMMs, but you are not required to do so.

A. (2 points) Implement the EM algorithm to estimate parameters (means, variances, weights) of a 1-dimensional GMM. Name your function "gmmest.py".

The function should be called like this:

```
def gmmest(X,mu_init,sigmasq_init,wt_init,its):
% Use numpy vectors/arrays.
% Input
%   - X          : N 1-dimensional data points (a 1-by-N vector)
%   - mu_init    : initial means of K Gaussian components
%                  (a 1-by-K vector)
%   - sigmasq_init: initial  variances of K Gaussian components
%                  (a 1-by-K vector
%   - wt_init    : initial weights of k Gaussian components
%                  (a 1-by-K vector that sums to 1)
%   - its        : number of iterations for the EM algorithm
%
% Output
%   - mu         : means of Gaussian components (a 1-by-K vector)
%   - sigmasq    : variances of Gaussian components (a 1-by-K vector)
%   - wt         : weights of Gaussian components (a 1-by-K vector, sums
%       to 1)
%   - L          : log likelihood
```

B. (2 points) Test your function by building a mixture model for each of the two classes in X_train. Choose an appropriate number of Gaussian components and initializations of parameters. Plot the data log-likelihood values for the first 20 iterations. Do you think your program has converged? Report the final values of the GMM parameters for class 1 and for class 2.

Hint: To pick good initializations, visualize the histogram of each class of data. The 'hist' function in matplotlib is a big help. So is the 'nonzero' function in numpy. Try this example code to get *started.* `(Y_test and X_test should be numpy arrays)`

```
import numpy as np
import matplotlib.pyplot as plt

class1 = X_test[np.nonzero(Y_test ==1)[0]]
class2 = X_test[np.nonzero(Y_test ==2)[0]]
bins = 50 # the number 50 is just an example.
plt.subplot(2,1,1)
plt.hist(class1, bins)
plt.subplot(2,1,2)
plt.hist(class2, bins)
plt.show()
```

C. (2 points) Implement a function to perform binary classification using your learned GMMs. Name your function "gmmclassify.py".

The function should be called as

```
def gmmclassify(X, mu1, sigmasq1, wt1, mu2, sigmasq2, wt2, p1):
% Use numpy vectors/arrays.
% Input
%   - X          : N 1-dimensional data points (a 1-by-N vector)
%   - mu1        : means of Gaussian components of the 1st class
%       (a 1-by-K1 vector)
%   - sigmasq1   : variances of Gaussian components of the 1st class
%       (a 1-by-K1 vector)
%   - wt1        : weights of Gaussian components of the 1st class
%       (a 1-by-K1 vector, sums to 1)
%   - mu2        : means of Gaussian components of the 2nd class
%       (a 1-by-K2 vector)
%   - sigmasq2   : variances of Gaussian components of the 2nd class
%       (a 1-by-K2 vector)
%   - wt2        : weights of Gaussian components of the 2nd class
%       (a 1-by-K2 vector, sums to 1)
%   - p1         : the prior probability of class 1.
```

D. (2 points) Run gmmclassify on X_test with the two GMMs you learned and the class prior probability p1 calculated from the training data. Report on the accuracy of your classifier (Just a single number, no need to do n-fold validation or statistics on this problem). Make a two-color histogram of the two classes in X_test, where the color indicates the ground-truth correct class for the data (see the example code in the hint above for a starting point). Then show classes learned by your GMM on this same plot by putting a colored marker (where the color indicates the class chosen by your classifier) on the horizontal axis for each classified data point.

2) About the math of GMMs (2 points)

A. (1 point) Assume I have a Gaussian Mixture Model (GMM) with K Gaussians. Assume that for every data point x in my data set X, I was told that exactly one Gaussian had generated it AND which Gaussian was the one that generated it. Would I be able to use a closed-form solution to find the parameters for the GMM that maximize the probability of the data without having to use Expectation Maximization? If so, say how. If not, say why not.

B. (1 point) Again assume I have a Gaussian Mixture Model (GMM) with K Gaussians. This time, I don't know which Gaussian generated any particular data point or even if only a single Gaussian was responsible for any data point. Must I use Expectation Maximization in this case? Give an explanation of why, or why not.