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.

Building a GMM classifier

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 homework, 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.

1. (2 points) Implement the EM algorithm to estimate parameters (means, variances, weights) of a 1-dimensional GMM.

We have provided starter code in `gmm_est.py` that looks like this:

```
def gmm_est(X, mu_init, sigmasq_init, wt_init, its):
"""
Input Parameters:
  - X           : N 1-dimensional data points (a 1-by-N np array)
  - mu_init     : initial means of K Gaussian components (a 1-by-K np array)
  - sigmasq_init: init variances of K Gaussian components (a 1-by-K np array)
  - wt_init     : init weights of k Gaussian components (a 1-by-K np array)
                  (sums to 1)
  - its         : number of iterations for the EM algorithm

Returns:
  - mu          : means of Gaussian components (a 1-by-K np array)
  - sigmasq     : variances of Gaussian components (a 1-by-K np array)
  - wt          : weights of Gaussian components (a 1-by-K np array, sums to 1)
  - L           : log likelihood
"""
```

At this point your code in this part should not output anything, it should just be the implementation of the EM algorithm. In part 2, you will use the `gmm_est()` function that you have written.

2. (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. **In your write up**, include a plot the data log-likelihood values for the first 20 iterations. Do you think your program has converged? **In your write up**, report the final values of the GMM parameters for class 1 and for class 2.

EECS 349 (Machine Learning) Homework 6

Your program should be runnable from the command line with the following command:

```
python gmm_est.py <path/to/gmm_test.csv>
```

In addition to your plots, your program should output the following, **in this exact format**, and nothing else:

```
>> python gmm_test.py gmm_test.csv
Class 1
mu = [  0.0  1.0]
sigma^2 = [ 2.0   3.0]
w = [ 4.0  5.0]

Class 2
mu = [ 0.0   1.0]
sigma^2 = [   2.0   3.0]
w = [ 4.0  5.0]
```

Note: These print statements have been provided for you in the starter code. [0.0 1.0], etc. are dummy values to give you an idea of the output format.

Additionally, **your plots should not pop up when running your program** (as they would when you call `plt.show()`), but rather they should be saved right next to your program (using `plt.savefig()`). Make sure to save them with names that make sense, like `likelihood_class1.png`.

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()
```

EECS 349 (Machine Learning) Homework 6

3. (2 points) Implement a function to perform binary classification using your learned GMMs. We have provided starter code in `gmm_classify.py`.

The function should be called as

```
def gmm_classify(X, mu1, sigmasq1, wt1, mu2, sigmasq2, wt2, p1):
"""
Input Parameters:
    - X         : N 1-dimensional data points (a 1-by-N numpy array)
    - mu1       : means of Gaussian components of the 1st class
                  (1-by-K1 np array)
    - sigmasq1 : variances of Gaussian components of the 1st class
                  (a 1-by-K1 np array)
    - wt1       : weights of Gaussian components of the 1st class
                  (1-by-K1 np array, sums to 1)
    - mu2       : means of Gaussian components of the 2nd class
                   (a 1-by-K2 np array)
    - sigmasq2 : variances of Gaussian components of the 2nd class
                  (a 1-by-K2 np array)
    - wt2       : weights of Gaussian components of the 2nd class
                  (a 1-by-K2 np array, sums to 1)
    - p1        : the prior probability of class 1.

Returns:
    - class_pred  : a numpy array containing results from the gmm classifier
                     (the results array should be in the same order as
                      the input data points)
"""
```

Your code should make a prediction for each data point in the numpy array X, put them in a numpy array with the same order, and return that numpy array containing the results. For instance, if I have two Guassians with Class 1 centered around 0.0, and Class 2 centered around 4.5, when I input a numpy array like this to my function:

```
[0.3, 0.5, 4.0, 5.1, 4.5, -0.1]
```

I would expect my function to return a numpy array like this:

```
[1, 1, 2, 2, 2, 1]
```

Where the ith value of the output array is the class of the ith value of the input array.

Your program should be runnable from the command line with the following command:

```
python gmm_classify.py <path/to/gmm_train.csv>
```

And it should output a list of the original data points that are in each class like this, **in this exact format**, and nothing else:

```
>> python gmm_classify.py gmm_train.csv
Class 1
[0.3, 0.5, -0.1, 0.0, 0.2, 0.0, 0.0, 0.1]

Class 2
[4.0, 5.1, 4.5, 4.6, 4.3, 4.5, 4.4, 4.5]
```

4. (2 points) Run gmm_classify on X_test with the two GMMs you learned and the class prior probability p1 calculated from the training data. **In your write up**, 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. **These plots should also be in your write up.**

Note: You may use your code from problem 3 do this problem, but make sure whatever edits you do, do not interfere with the expected output of gmm_classify from problem 3 (this includes popping up plots with matplotlib!).

About the math of GMMs

5. (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.

6. (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.