Autonomous Agents

Assignment 2: Single Agent Learning 5 October, 2012

By:
Paris Mavromoustakos
Georgios Methenitis
Marios Tzakris

Introduction

A mixture distribution is the probability distribution of a random variable whose values can be interpreted as being derived from an underlying set of random variables. In the above figure, it is shown that a **latent variable** z underlies between class C and data-point x, thus latent variable z controls which data-points belong to which mixture element.

Exercise 1

First, we loaded the given data files `banana.mat`, and, `spiral.mat`, each one of them contains two-dimensional data points, represent two different classes. The training set, which we used in order to train our classifier consists of 75% of data points from class A, appended to 75% of data points from class B, while the test set contains the remaining 25% of both classes A and B. Figure 1, shows the 2-D representation of the two classes of data-points.

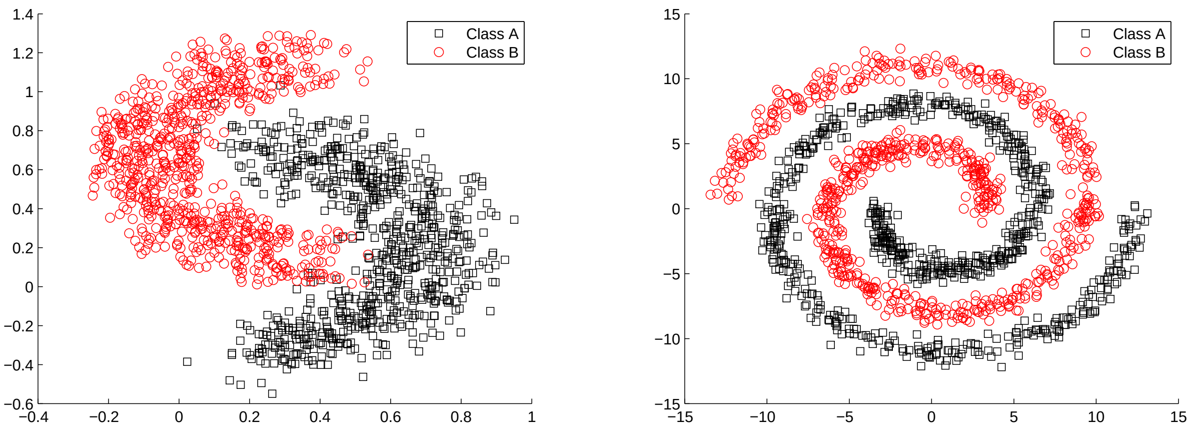

Figure 1: Depiction of the 2-D data-points of the two given classes for the `banana.mat` (left) and `spiral.mat` (right).

Looking into these data-points, we can figure out that the distribution of the data in both cases is following a "banana" and a spiral shape respectively. Therefore, a single 2-D Gaussian distribution could not be able to describe the class conditional probabilities well. For this reason, we do not expect the model to perform accurately. In order to perform the training, we need the prior probabilities, the mean μ_k of each class and the covariance matrix Σ_k for each class $k \in \{A, B\}$. We set the prior probabilities 0.5 as the number of the data for class A and B is the equal. Next, we can directly extract the mean and and the covariance for each class directly from the training data. The Matlab code for that is given below:

```
[A, B] = load( data_points )
TrainingSetA = A(1:length(A)*3/4,:)
TrainingSetB = B(1:length(B)*3/4,:)
TrainingSet = [TrainingSetA; TrainingSetB]

prior_probability_A = size(A,1)/(size(A,1)+size(B,1))
prior_probability_B = size(B,1)/(size(A,1)+size(B,1))

for i=1:size(TrainingSetA,1)
   sumAx = sumAx + TrainingSetA(i,1);
   sumAy = sumAy + TrainingSetA(i,2);
   sumBx = sumBx + TrainingSetB(i,1);
   sumBy = sumBy + TrainingSetB(i,2);
end

mean_A = [sumAx sumAy]/size(TrainingSetA,1);
mean_B = [sumBx sumBy]/size(TrainingSetA,1);

covariance_A = cov(TrainingSetA);
covariance_B = cov(TrainingSetB);

pA = mvnpdf(TestSet,mean_A,covariance_A);
pB = mvnpdf(TestSet,mean_B,covariance_B);

PostA = (pA .* prior_probability_A) ./ (pA .* pr_classA + pB .* pr_classB);
PostB = (pB .* prior_probability_B) ./ (pA .* pr_classA + pB .* pr_classB);
```

After the classification process we got the following confusion matrices and error rates for the our classifier:

Banana set

$$Confusion\ Matrix = \begin{bmatrix} 226 & 24 \\ 26 & 224 \end{bmatrix},$$
$$error\ rate = 0.1$$

Spiral set

$$Confusion\ Matrix = \begin{bmatrix} 174 & 76 \\ 89 & 161 \end{bmatrix},$$
$$error\ rate = 0.33$$

As we said before, the way that we tried to perform classification in this type of data, did not perform well, mainly because, the shape of the data, and the fact that, data-points from the one class are located inside the other class, make it impossible for a simple 2-D Gaussian distribution to represent them.

Exercise 2

To overcome the problem that we came across in the previous exercise, we had to use a different model that, the class-conditional distributions are modelled by a mixture of Gaussians distributions and not only by one as we did in the previous approach.

Using the given function, `em_mog(X,C,verbose)`, for the training set for both classes A and B, with input arguments the training set and then the number of mixture components we wanted to use. We followed this procedure for both the training sets of classes A and B. Each iteration of the EM algorithm, calls two functions E-step and M-step. E-step is responsible of updating the responsibilities for each i-component $p(\mathbf{z}|\mathbf{t}, \theta^{\mathbf{old}})$. For each component the update rule is:

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x_n}|\mu_\mathbf{k}, \sum_\mathbf{k})}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x_n}|\mu_\mathbf{j}, \sum_\mathbf{j})}$$

The Matlab code for the E-step of the EM algorithm is the following:

```
L = length(MOG);
N = length(X(:,1));
Q = zeros(N, L);
total = zeros(N, 1);

for i = 1:L
    prob = MOG{i}.PI * mvnpdf(X, MOG{i}.MU, MOG{i}.SIGMA);
    Q(:,i) = prob;
    total = total + prob;
end

Q = Q ./ repmat(total, [1 L]);
LL = sum(log(total));
```

Next, we had to implement the M-step of the algorithm. After the update of the responsibilities for each component, we have to re-estimate their parameters:

$$\pi_k^{new} = \frac{N_k}{N} \ (1)$$

$$\mu_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(Z_{nk}) x_n \ (2)$$

$$\sum_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(Z_{nk})(x_n - \mu_k^{new})(x_n - \mu_k^{new})^T \ (3)$$

The Matlab code for the M-step of the EM algorithm is the following:

```
L = length(MOG);
N = length(X(:,1));

for i = 1:L

   N_i = sum(Q(:,i));

   MOG{i}.PI = N_i / N; (1)
   MOG{i}.MU = sum(repmat(Q(:,i),[1 2]) .* X(:,1:2)) / N_i; (2)
   temp = (repmat(Q(:,i)', [2 1]) .* ((X(:,1:2) - repmat(MOG{i}.MU, [N 1]))') * (X(:,1:2)
          - repmat(MOG{i}.MU, [N 1]))); (3)

   temp = temp / N_i;

   if cond(temp) < 10^10
      MOG{i}.SIGMA = temp;
   end
end
```

We also included a check in our M-step to avoid singularities. If c becomes larger than the number 10^{10}, we do not perform the update. After these two steps of the EM algorithm, we check the convergence of the parameters. To realize when the algorithm converges, we just check how much change occurred in θ variable after the E-step.

"Banana" data set

As we expected, testing this classifier gave us better results than the one at the previous exercise. The number of components was an important factor on how the classifier would have performed. In Figure 2, you can see how the Gaussian components are formed throughout training process after the converge of the algorithm.

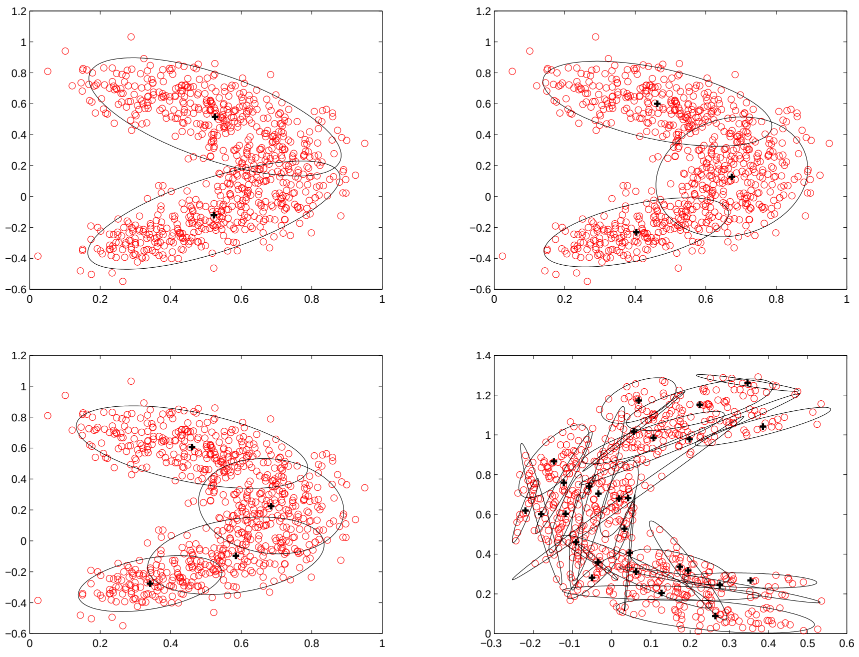

Figure 2: Illustration of the converged parameters, on the "Banana" dataset, using **2** (top-left), **3** (top-right), **4** (bottom-left) and, **30** (bottom-right).

In general, even in the training phase of our classifier we want to keep it simple so the less components the fewer time it takes for the EM algorithm to converge. To test this, we performed a test using different number of components and evaluating the error rate. Figure 5, presents the error rate in condition to the number of mixture components. Looking at this figure we can realize that the gain in the error rate, for 2 to 30 components, of our classifier is not big enough to convince us to use more than 2, 3, or 4 components. In contrary that the lower error rate was when we used 19 components, it took to the algorithm 650 iterations to converge for the training sets of both classes. In contrast, for only 3 components, algorithm needed 155 iterations. This big difference in the number of iterations led us to the conclusion that 4 is a good number of components for the "banana" set of data-points.

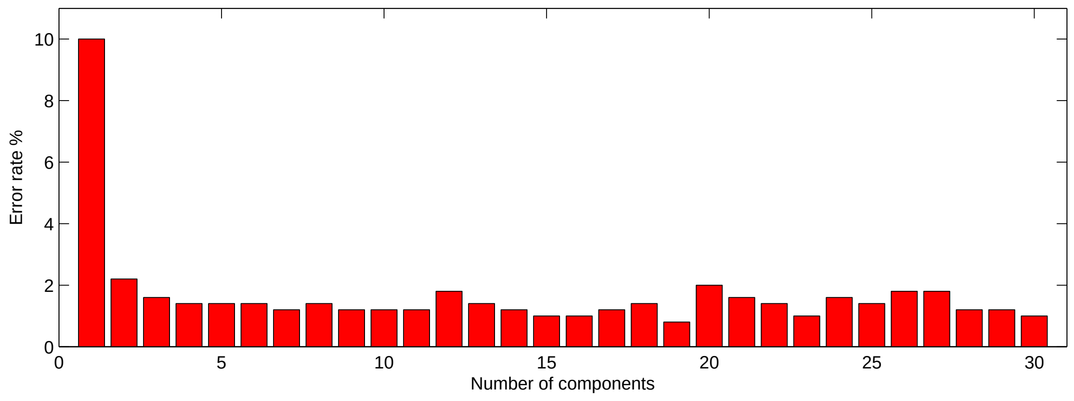

Figure 3: Error rate for EM algorithm using different number of componentson the "Banana" data.

"Spiral" data set

As before, We train our classifier in the "Spiral" training set. In Figure 4, you can see how the Gaussian components are formed after the converge of the algorithm.

In contrary with the previous "Banana" dataset, after performing the same test as before using

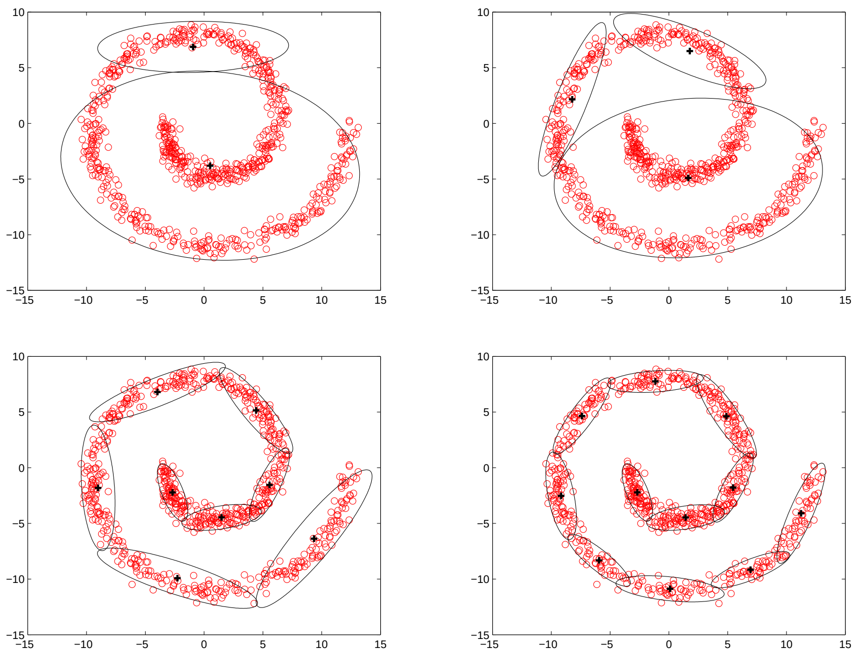

Figure 4: Illustration of the converged parameters, on the "Spiral" dataset, using **2** (top-left), **3** (top-right), **8** (bottom-left) and, **11** (bottom-right).

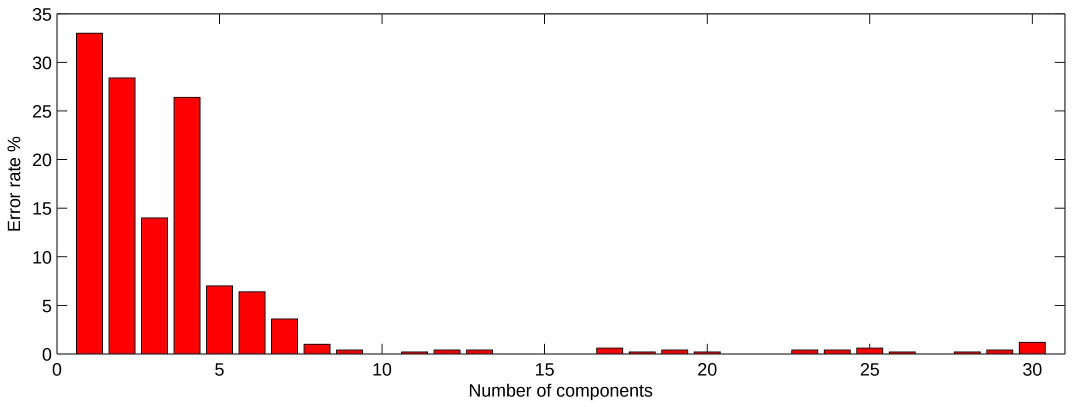

Figure 5: Error rate for EM algorithm using different number of componentson the "Spiral" data.

different number of components and evaluating the error rate on the "Spiral" dataset, we came up with different results. This happened due to the shape of the data, and the fact that the spiral shape cannot be represented by only two Gaussian distributions. We got appropriate, for a classifier, results after 7 we used more than 7 components. In fact, using 10 components gave us an error-free classifier, which shows the power of the EM algorithm especially in a such difficult dataset in which classes' data-points in not easy to be distinguished by other classifiers which we saw in previous labs or the first exercise of this lab.

Exercise 3

First we implemented the logsumexp function that uses an iteration in order to compute the log sum of two or more arguments. It takes as input a list of logarithms and computes the logsumexp of the elements at indices i and $i+1$. Then we advance i by 1 until the end of the list:

```
function logsum = logsumexp (X)

    n = size(X,2);

    for i=1 : (n-1)

        lnpa = X(:,i);
        lnpb = X(:,i+1);

        maxv = max(lnpa,lnpb);
        minv = min (lnpa,lnpb);

        X(:,i+1) = (maxv + log(ones(size(lnpa,1),1)+exp(minv-maxv)));
    end

    logsum = X(:, end);

end
```

After that we checked our implementation by computing $ln[p(a) + p(b)]$ when $lnp(a) = -1000$ and $lnp(b) = -1001$ and the result was -999.6867

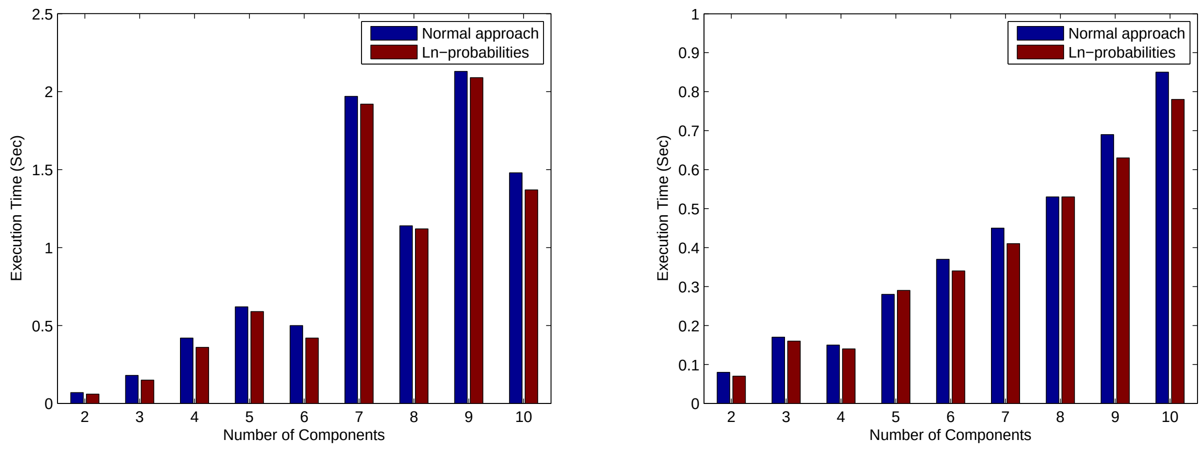

Figure 6:

Finally we converted the E-step from the previous exercise:

```
function [Q LL] = mog_E_step_ex3(X,MOG)
    %Number of components
    N = length(X(:,1));
    K = length(MOG);
    Q = zeros(N, K);
    for k = 1:K
        Q(:,k) = log(MOG{k}.PI) + lmvnpdf(X, MOG{k}.MU, MOG{k}.SIGMA);
    end
    Qsum = logsumexp(Q);

    for k=1:k
    Q(:,k) = Q(:,k)-Qsum;
    end
    Q=exp(Q);
    LL = sum(Qsum);
```

Although the new implementation is numerically more stable, it is not more accurate than the previous one, as we get the exactly same error rate. The only thing that we observed and we can mention as an improvement is the run-time. In the following array are presented the parameters and the run time of both implementations:

Conclusion