MTAT.03.227 Machine Learning (Spring 2012)
Excercise session: K-means and Gaussian mixture models

Meelis Kull
Exercise session: April 10, 2012
Homework deadline: April 17, 2012 at 12:15 EET

Important notes:

1) For homework provide a report in the PDF-format. The report should include R code and explanatory figures, formulas and text.
2) The code of every programming task must be accompanied with a small test code showing that the code works correctly at least on one data set. For testing you are encouraged to use the data sets generated in Task 1. But any other data with generating code included can also be used.
3) For simplicity all the tasks are about 1-dimensional data with two clusters, but can be extended to higher dimensions and more clusters relatively easily.
4) The points for tasks sum up to 11, but homework results above 10 will be awarded 10 points.

Task 1 (0.5 points): Write a function to generate 1-dimensional data from a Gaussian mixture model (GMM) of two Gaussians:

```
gmm2 <- function(n=1000,mu=c(0,1),sigma2=c(1,1),pi=c(0.5,0.5))
```

Parameter `n` stands for the number of data points to generate, `mu` is a 2-element list with the means for the two Gaussians, `sigma2` are the two variances and `pi` are the mixing probabilities (the probabilities of a data point to be generated according to one or the other Gaussian).

Generate four datasets for testing this and all other programming tasks:

```
test.data.A = gmm2(100,c(0,100),c(1,10),c(0.9,0.1))
test.data.B = gmm2(10000,c(0,2),c(1,1),c(0.5,0.5))
test.data.C = gmm2(10000,c(0,2),c(1,1),c(0.1,0.9))
test.data.D = gmm2(10000,c(0,3),c(1,2),c(0.1,0.9))
```

Task 2 (1 point): Implement 1-dimensional k-means with $k = 2$. The algorithm should iterate until the cluster contents do not change.

```
kmeans2 <- function(data) {
  centers = c(0,1)
  cluster = rep(1,times=length(data))
  ### here goes your k-means code ###
  result = list(centers,cluster)
  names(result) = c("centers","cluster")
  result
}
```

For the meaning of result vectors refer to the R standard K-means implementation (`kmeans`).

Compare the locations of cluster centers with the R standard K-means.

Task 3 (0.5 points): Write a function `kmeans.viz` to visualize the result of K-means. The visualization should include at least the location of cluster centers and histograms of both clusters. It should also be possible to add the *true centers* to the visualization if the data was generated with GMM.

Visualize the result of K-means and compare the locations of cluster centers with the true centers. What do you observe and why this happens? Check your hypothesis with increasing data size.

Task 4 (1 point): Derive formulas to fit a Gaussian with unit variance (`sigma2=1`) on 1-dimensional data (0.5 points). Define a function fitting the data vector and returning the mean `mu` (0.5 points):

```
fit.gaussian <- function(data)
```

Task 5 (0.5 points): Define a function which fits two Gaussians on the two clusters from the k-means result separately and returns the vector of two means.

```
fit.clusters <- function(data,kmeans.result)
```

Compare the obtained means of Gaussians with the cluster centers as obtained by k-means. Explain what you see and why.

Task 6 (1 point): Write a function to calculate the *responsibility* of each data point with respect to each Gaussian in a Gaussian mixture. The responsibility is the probability (density) of a Gaussian generating this particular data point, normalized (multiplied by a constant) so that for each data point the responsibilities to all the clusters sum up to one. *E.g.*, if one Gaussian generates the point with probability (density) 0.1 and the other with 0.3, then the responsibilities should be 0.25 and 0.75. If the mixture is uneven (`pi` is not `c(0.5,0.5)`), then the probabilities should be multiplied by the respective component frequency (`pi[1]` or `pi[2]`) before normalizing.

```
responsibility <- function(data,mu=c(0,1),sigma2=c(1,1),pi=c(0.5,0.5))
```

Task 7 (1 point): K-means puts equal weight to each data point in calculating the cluster means. Suppose we would like to put more weight on the points which are between the clusters, in order to move cluster centers closer to each other. Let us define a weight for a data point as `w=round(10/max(r1, r2))` where `r1` and `r2` are the responsibilities of this point to the two Gaussians of a mixture. This means that a point with responsibilities 0.3 and 0.7 gets the weight `round(10/0.7)=14`. One way to run weighted K-means is to replicate the data points, such that some points have more and some less replicates (the number of replicates is the weight `w`), and then run the usual K-means on the replicated data. Define a function for doing this.

```
kmeans.biased <- function(data) {
  ### 1) run your original K-means implementation         ###
  ### 2) fit the unit-variance Gaussians to both clusters ###
  ### 3) calculate responsibilities                       ###
  ### 4) calculate weights for all points                 ###
  ### 5) replicate the data according to the weights      ###
  ### 6) run your original K-means on the replicated data ###
}
```

Note that the given formula for calculating `w` is absolutely arbitrary and has no any deeper justification or meaning.

Task 8 (1 point): Combine your `kmeans2` and `kmeans.biased` codes to take the weights into

account without any need for data replication. Note that the code should give exactly the same result as in the previous task.

```
kmeans.biased2 <- function(data) {
  ### 1) run your original K-means implementation         ###
  ### 2) fit the unit-variance Gaussians to both clusters ###
  ### 3) calculate responsibilities                       ###
  ### 4) calculate weights for all points                 ###
  ### 5) do modified K-means to account for the weights   ###
}
```

Task 9 (1 point): Implement a single iteration of soft clustering by modifying the weighted K-means such that every point contributes to each cluster. This means that for each cluster there is a different weight vector. For weights use the responsibility vectors. Note that the weights should be applied in the calculation of cluster means with exactly the same way as in the previous task, except now the weights are not restricted to integers.

```
soft.iteration <- function(data) {
  ### 1) run your previous K-means implementation         ###
  ### 2) fit the unit-variance Gaussians to both clusters ###
  ### 3) calculate responsibilities                       ###
  ### 4) use responsibilities for calculating means as
  ###    in kmeans.biased2                                ###
}
```

Visualize the result with `kmeans.viz`. Compare the locations of centers with true centers.

Task 10 (1 point): Implement EM algorithm applied for GMM with two unit variance equally populated clusters. For this you need to run many iterations from the previous tasks until convergence. Instead of initializing with the K-means you can use `c(0,1)` as initial Gaussian centers. Iterate until the means of Gaussians obtained at two consecutive iterations differ by less than epsilon.

```
em.mu <- function(data,epsilon=1e-10) {
  ### 1) initialize means e.g. c(0,1)                    ###
  ### 2) iterate the following until convergence         ###
  ### 3) calculate responsibilities                      ###
  ### 4) use responsibilities for calculating new means  ###
}
```

Compare the locations of means with true centers.

Task 11 (0.5 points): What happens with `em.mu` if one component of the mixture has 10 times more points than the other? Why?

Task 12 (1 point): Introduce the probabilities `pi` for the components of the mixture to the EM-algorithm `em.mu` and define `em.mu.pi`. You should include these probabilities in the calculation of responsibilities. In each iteration the new value of `pi` for a component should be obtained by adding the responsibilities of all data points with respect to this cluster.

Task 13 (1 point): Introduce `sigma2` to the EM-algorithm `em.mu.pi` to obtain `em.gmm`. The values `sigma2` should be included in the calculation of responsibilities. In each iteration the new value of `sigma2` for a component should be obtained by summing up the squared deviations from the mean weighted by the responsibilities.