CSE847 HW3

Javen W. Zamojcin
zamojci1@msu.edu

2024, Novemer 27

1 Gaussian Mixture Model And EM Algorithm

1.1 Parameters

The number of independent parameters in a given GMM is calculated as $3k-1$.

$3k - 1 = 3(2) - 1 = 6 - 1 = 5$

1.2 EM Algorithm

See Figure 1 for EM algorithm computation steps. The included file `HW3_Q1.pdf` contains the Python implementation code from scratch.

2 Graphical Models

2.1 Joint Distribution

$P(Y, X1, X2, X3, X4, X5, X6) =$

$P(Y|X2, X5) * P(X1) * P(X2) * P(X3|X2, X6) * P(X4|X3) * P(X5|X1, X6) * P(X6)$

3 K-Means

3.1 K-means Steps

Figure 2 displays the distances calculated for each point-centroid pair for each iteration, the cluster membership for each point for each iteration, and the cluster centroids for each iteration.

```
Gaussian Density (x=-67, k=0) = 0.019947114020071634, (x=-67, k=1) = 2.07929948955756 53e-20
Gaussian Density (x=-67, k=0) = 0.019947114020071634, (x=-67, k=1) = 2.07929948955756 53e-20
Gaussian Density (x=-48, k=0) = 0.0032807907387338298, (x=-48, k=1) = 1.10399498156856 95e-13
Gaussian Density (x=-48, k=0) = 0.0032807907387338298, (x=-48, k=1) = 1.10399498156856 95e-13
Gaussian Density (x=6, k=0) = 5.346918935770819 4e-14, (x=6, k=1) = 0.00394750791504470 7
Gaussian Density (x=6, k=0) = 5.346918935770819 4e-14, (x=6, k=1) = 0.00394750791504470 7
Gaussian Density (x=8, k=0) = 1.217160266514504 7e-14, (x=8, k=1) = 0.00554604173397277 8
Gaussian Density (x=8, k=0) = 1.217160266514504 7e-14, (x=8, k=1) = 0.00554604173397277 8
Gaussian Density (x=14, k=0) = 1.1294047015771514e-16, (x=14, k=1) = 0.012098536225957168
Gaussian Density (x=14, k=0) = 1.1294047015771514e-16, (x=14, k=1) = 0.012098536225957168
Gaussian Density (x=16, k=0) = 2.1908197177546787e-17, (x=16, k=1) = 0.014484577638074137
Gaussian Density (x=16, k=0) = 2.1908197177546787e-17, (x=16, k=1) = 0.014484577638074137
Gaussian Density (x=23, k=0) = 5.139886785834457e-20, (x=23, k=1) = 0.01984762737385059
Gaussian Density (x=23, k=0) = 5.139886785834457e-20, (x=23, k=1) = 0.01984762737385059
Gaussian Density (x=24, k=0) = 2.079299489557565 3e-20, (x=24, k=1) = 0.019947114020071634
Gaussian Density (x=24, k=0) = 2.079299489557565 3e-20, (x=24, k=1) = 0.019947114020071634

Responsibility (x=-67, k=0) = 1.0
Responsibility (x=-67, k=1) = 1.0424061783901 6e-18
Responsibility (x=-48, k=0) = 0.9999999999663497
Responsibility (x=-48, k=1) = 3.365027121350292 5e-11
Responsibility (x=6, k=0) = 1.354504930900902e-11
Responsibility (x=6, k=1) = 0.9999999998986455
Responsibility (x=8, k=0) = 2.194646785753537 7e-12
Responsibility (x=8, k=1) = 0.9999999999978053
Responsibility (x=14, k=0) = 9.335052443402414e-15
Responsibility (x=14, k=1) = 0.9999999999999907
Responsibility (x=16, k=0) = 1.512518882149445 8e-15
Responsibility (x=16, k=1) = 0.9999999999999984
Responsibility (x=23, k=0) = 2.589673157914 23e-18
Responsibility (x=23, k=1) = 1.0
Responsibility (x=24, k=0) = 1.042406178390 16e-18
Responsibility (x=24, k=1) = 1.0

Weight (k=0) = 0.2499999999977625
Mean (k=0) = -57
Variance (k=0) = 90
Weight (k=1) = 0.7500000000022375
Mean (k=1) = 15
Variance (k=1) = 46
```

Figure 1: Q1.2 EM Calculations

3.2 Potential Function

Figure 3 displays the loss (Euclidean distance) calculated for each point in each cluster, the cluster membership for each iteration, and the total loss calculated for all the final clusters.

The total K-means Loss I calculated was 6.0, but I wasn't completely sure if the individual point-cluster distances should be squared or not. If so, the total loss would be 10.0.

3.3 Implementation

3.3.1 Pseudo Code

My implementation begins with defining a `Cluster` class, which contains a `centroid` tuple variable, and a `points` variable which is a list of tuple points. `Cluster` also defined a `update_centroid()` function which recalculates the `centroid` by simply calculating the mean of the all the assigned points in `points`.

The `euclidean_distance()` function takes two numpy array points as parameters and calculates distance as $\sum(a-b)^2$.

The `calculate_loss()` function takes a list of clusters as input, and for each assigned point in each cluster object, calculates the euclidean distance between the point and the respective cluster centroid and sums the total distance as the returned loss value.

The `assign_points()` function takes a list of clusters and the list of all samples as input. It first clears any existing cluster point assignments, then cal-

```
i=0:

x=[3 3], centroid=[6 5], distance=13.0
x=[7 9], centroid=[6 5], distance=17.0
x=[9 7], centroid=[6 5], distance=13.0
x=[5 3], centroid=[6 5], distance=5.0

x=[3 3], centroid=[6 6], distance=18.0
x=[7 9], centroid=[6 6], distance=10.0
x=[9 7], centroid=[6 6], distance=10.0
x=[5 3], centroid=[6 6], distance=10.0

i=1:

x=[3 3], centroid=[4. 3.], distance=1.0
x=[7 9], centroid=[4. 3.], distance=45.0
x=[9 7], centroid=[4. 3.], distance=41.0
x=[5 3], centroid=[4. 3.], distance=1.0

x=[3 3], centroid=[8. 8.], distance=50.0
x=[7 9], centroid=[8. 8.], distance=2.0
x=[9 7], centroid=[8. 8.], distance=2.0
x=[5 3], centroid=[8. 8.], distance=34.0

i=2:

x=[3 3], centroid=[4. 3.], distance=1.0
x=[7 9], centroid=[4. 3.], distance=45.0
x=[9 7], centroid=[4. 3.], distance=41.0
x=[5 3], centroid=[4. 3.], distance=1.0

x=[3 3], centroid=[8. 8.], distance=50.0
x=[7 9], centroid=[8. 8.], distance=2.0
x=[9 7], centroid=[8. 8.], distance=2.0
x=[5 3], centroid=[8. 8.], distance=34.0
```

iter	c0	c1	[3 3]	[5 3]	[7 9]	[9 7]
0	[6 5]	[6 6]	0	0	1	1
1	[4. 3.]	[8. 8.]	0	0	1	1
2	[4. 3.]	[8. 8.]	0	0	1	1

Figure 2: Q3.1 Cluster Membership and Centroids

culates a new distance matrix between each cluster centroid and all sample points. It then calculates the minimum cluster index through the distance matrix (`np.argmin()`) for each point, assigning each point to its respective minimum distance cluster.

The `update_centroids()` function takes a list of clusters as input. It first performs the sub-routine of detecting and fixing any empty clusters. If an empty cluster is found, a copy of all clusters is made and sorted by cluster size to find the largest cluster. The largest cluster is then split in half (non-randomly), with the first half of points being re-assigned to the empty cluster. After this

```
x=[3 3], centroid=[4. 3.], distance=1.0
x=[5 3], centroid=[4. 3.], distance=1.0

x=[7 9], centroid=[8. 8.], distance=2.0
x=[9 7], centroid=[8. 8.], distance=2.0
```

iter	c0	c1	[3 3]	[5 3]	[7 9]	[9 7]
0	[6 5]	[6 6]	0	0	1	1
1	[4. 3.]	[8. 8.]	0	0	1	1
2	[4. 3.]	[8. 8.]	0	0	1	1

```
loss=6.0
```

Figure 3: Q3.2 K-Means Loss Function

sub-routine, each cluster then has its `update_centroid()` function called to recalculate its centroid point.

Finally, the `k_means()` function accepts all sample points, an integer value `K`, a number of training iterations, and optionally a list of predefined clusters. If the clusters are not already defined, a new list of size `K` is initialized with a random sample point selected as the initial centroid for each one. Then, the K-means loop is ran for each iteration: first, assign all points to all current clusters; secondly, generate the history (logging purposes) of each point and its cluster assignment for the iteration; and finally, update the centroids with the next point assignments. After this loop is finished, the final loss is calculated and returned.

3.3.2 Results

Figure 4 displays the total calculated loss (potential function) for each value of K. I wouldn't necessarily select the largest value of K (8), which has the lowest total loss. A K value of 6 seems to be good, as there are diminishing returns past this point. The higher value of K, the more complexity the model has and the less efficient training is.

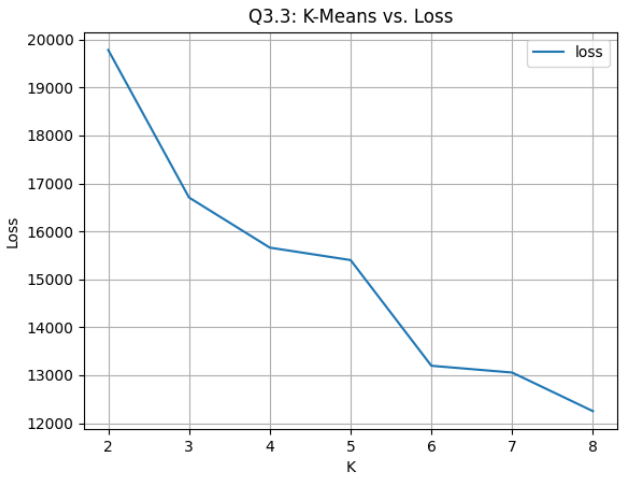

Figure 4: Q3.3 L(K) vs. K Value