CS498: Applied Machine Learning **Spring 2018**

Machine Problem 7 - EM on GMM In Matrix form

Professor *David A. Forsyth* *Ehsan Saleh*

Definitions

- N: Number of data points
- K: Number of clusters
- d: Dimension of the data space
- $\mathbf{u}^T$: The transpose of $\mathbf{u}$
- X: The data matrix

$$X = \begin{bmatrix} \mathbf{x}_1^T \\ \hline \mathbf{x}_2^T \\ \hline \cdots \\ \hline \mathbf{x}_N^T \end{bmatrix}$$

- $\mathbf{x_i}$: The vector of data point i, with a dimension of d rows and one column
- $\mu_\mathbf{j}$: The vector of centroid j, with a dimension of d rows and one column
- μ: The means matrix

$$\mu = \begin{bmatrix} \mu_1^T \\ \hline \mu_2^T \\ \hline \cdots \\ \hline \mu_K^T \end{bmatrix}$$

- $A \circ B$: The Hadamard product of matrices A and B (i.e. the element-wise multiplication of A and B matrices).
- $A \oslash B$: The Hadamard division of matrices A and B (i.e. the element-wise division of A over B matrices).
- $A \cdot B$: The matrix multiplication of A and B matrices.
- $A_{m \times n}$: Some matrix A having m rows and n columns.
- $\mathbf{1}_{m \times n}$: matrix of all one elements, with m rows and n columns.
- W: The posterior probability matrix with elements of $w_i j$ as described in the textbook.
- π: The prior probability vector of π_j values as described in the textbook.

$$\pi = \begin{bmatrix} \pi_1 \\ \pi_2 \\ \cdots \\ \pi_K \end{bmatrix}$$

- $\exp(A)$: The element-wise exponentiation of matrix A.
- $\log(A)$: The element-wise logarithm of matrix A.

The E-Step

Let's define a matrix H, whose dimension is N rows and d columns

$$H[i,j] = -\frac{1}{2}\Big[(\mathbf{x}_i - \mu_j)^T(\mathbf{x}_i - \mu_j)\Big] = -\frac{1}{2}\Big[\mathbf{x}_i^T\mathbf{x}_i + \mu_j^T\mu_j - 2\mathbf{x}_i^T\mu_j\Big]$$

We will try to compute the H matrix through matrix operations. In order to do so, we will define 3 matrices.

$$H_1[i,j] = \mathbf{x}_i^T\mathbf{x}_i$$

$$H_2[i,j] = \mu_j^T\mu_j$$

$$H_3[i,j] = \mathbf{x}_i^T\mu_j$$

Then

$$H = -\frac{1}{2}(H_1 + H_2 - 2H_3)$$

Now, let's try to compute each of the 3 matrices with common matrix operations.

H_1 matrix

$$H_1[i,j] = \mathbf{x}_i^T\mathbf{x}_i$$

Steps for H_1

1. $$U = \Big[X \circ X\Big]_{N\times d}$$

   This will give you a squared data matrix with N rows, and d columns.

2. Now we need to sum over the rows of U in order to do compute the $\mathbf{x}_i^T\mathbf{x}_i$ values.

   Therefore,

   $$V_{N\times 1} = U_{N\times d}\cdot\mathbf{1}_{d\times 1}$$

   will give you this summation vector. It has a column shape (i.e. it has N rows, and one column).

3. Now you need to repeat the V column K times along the column axis. One way to do so, is to multiply the V result with a right-hand$\mathbf{1}_{1\times K}$ matrix. In other words

   $$H_1 = V_{N\times 1}\cdot\mathbf{1}_{1\times K}$$

In order to do a recap:

$$H_1 = \Big[X \circ X\Big]\cdot\mathbf{1}_{d\times 1}\cdot\mathbf{1}_{1\times K}$$

Which simplifies to

$$H_1 = \Big[X \circ X\Big]\cdot\mathbf{1}_{d\times K}$$

Note 1: Is there a redundancy of computation happening in this process? Can you do this more efficiently? I mean, the repetition step could be done without any computation if you think about it...

Note 2: Again,... you may be able to speed up a little bit by doing the summation step without matrix multiplication. You'll have to figure out how.

Disclaimer about the note: No guarantees about speedups. It can really depend on the programming language you use. Matlab is too sensitive to non-matrix operations, while python seems less sensitive. You'll have to compare different methods, and see which is faster if you'd like to get better speed :).

H_2 matrix

$$H_2[i,j] = \mu_j^T \mu_j$$

Steps for H_2

1. $$A = \Big[\mu \circ \mu\Big]_{K \times d}$$
   This will give you a squared means matrix with K rows, and d columns.
2. Now we need to sum over the rows of A in order to do compute the $\mu_j^T \mu_j$ values.

   Therefore,
   $$B_{K\times 1} = A_{K\times d} \cdot \mathbf{1}_{d\times 1}$$
   will give you this summation vector. It has a column shape (i.e. it has N rows, and one column).
3. Now you need to first transpose the B column to get a row matrix, and then repeat the B column N times along the row axis. One way to do so, is to multiply the B^T result with a left-hand $\mathbf{1}_{N\times 1}$ matrix. In other words
   $$H_2 = \mathbf{1}_{N\times 1} \cdot B_{K\times 1}^T$$

In order to do a recap:

$$\begin{aligned} H_2 &= \mathbf{1}_{N\times 1} \cdot [\Big[\mu \circ \mu\Big] \cdot \mathbf{1}_{d\times 1}]^T \\ &= \mathbf{1}_{N\times 1} \cdot \mathbf{1}_{d\times 1}^T \cdot \Big[\mu \circ \mu\Big]^T \\ &= \mathbf{1}_{N\times 1} \cdot \mathbf{1}_{1\times d} \cdot \Big[\mu \circ \mu\Big]^T \\ &= \mathbf{1}_{N\times d} \cdot \Big[\mu \circ \mu\Big]^T \end{aligned}$$

Note 1: Again,... is there a redundancy of computation happening in this process? Can you do this more efficiently? I mean, the repetition step could be done without any computation if you think about it...

Note 2: Again,... you may be able to speed up a little bit by doing the summation step without matrix multiplication. You'll have to figure out how.

Disclaimer about the notes: No guarantees about speedups. It can really depend on the programming language you use. Matlab is too sensitive to non-matrix operations, while python seems less sensitive. You'll have to compare different methods, and see which is faster if you'd like to get better speed :).

H_3 matrix

$$H_3[i,j] = \mathbf{x}_i^T \mu_j$$

Well, This is pretty easy.

$$H_3 = X \cdot \mu^T$$

Final H

$$H = -\frac{1}{2}(H_1 + H_2 - 2H_3)$$

Therefore,

$$H = -\frac{1}{2}\left(\left[X \circ X\right] \cdot \mathbf{1}_{d\times K} + \mathbf{1}_{N\times d} \cdot \left[\mu \circ \mu\right]^T - 2X \cdot \mu^T\right)$$

Note: Some languages let you add a column matrix (i.e. with N rows and one column) to a row matrix (i.e. with one row and K columns), and get a populated matrix (i.e. with N rows and K columns). If you could do this, you may be even able to skip the repetition parts of H_1 and H_2, and just add a row matrix to a column matrix, and hopefully save some time! Try to do this wisely, if possible :)

W matrix

Let's try to compute the nominator of the w_ij elements,

$$W[i,j] = w_{ij} \propto \exp(H[i,j]) * \pi_j$$

Let's define a P matrix here that could help us:

$$P[i,j] = \pi_j$$

You can compute the P matrix using matrix multiplication (as I'm giving in the following), or even do some sort of repetition to try to minimize the computation cost (you'll have to think about how to do the repetition yourself.)

$$P_{N\times K} = \mathbf{1}_{N\times 1} \cdot [\pi^T]_{1\times K}$$

Or said easier:

$$P = \mathbf{1}_{N\times 1} \cdot \pi^T$$

So:

$$W[i,j] = w_{ij} \propto \exp(H[i,j]) * P[i,j]$$

Or equivalently:

$$W \propto \exp(H) \circ P$$

Let's give $\exp(H) \circ P$ a name:

$$E = \exp(H) \circ P$$

Now, in we have to find a matrix for the denominator term. Let's name it F. For computing F, we will have to sum over the columns of E, and then repeat the summation over the column axis. I will do both at once:

$$F = \mathbf{1}_{K\times 1} \cdot \mathbf{1}_{1\times N} \cdot E_{N\times K} = \mathbf{1}_{K\times N} \cdot E$$

$\mathbf{1}_{K\times 1}$ is for the repetition, and $\mathbf{1}_{1\times N}$ is for summation. Again, repetitions may be doable without matrix multiplication, and probably you could speed this up by doing summations and repetitions separately using predefined functions in your programming language.

In order to recap now:

$$W = E \oslash F$$

which means

$$W = \Big[\exp(H) \circ P\Big] \oslash \Big[\mathbf{1}_{K\times N} \cdot (\exp(H) \circ P)\Big]$$

Question: Can we please do these computations in the log space? I'm really having numerical stability issues...

Answer: Sure, let's do that:

$$\log(W) = \log(E) - \log(F)$$

$$\log(E) = H + \log(P)$$

- $\log(P)$ can be easily obtained from $\log(\pi)$ values. you'll have to take care of the repetition step.
- Once you have obtained $\log(E)$, you can easily compute $\log(F)$ using some sort of $log(sum(exp))$ function. You just have to apply the $log(sum(exp))$ function over each row of $\log(E)$ matrix, and then do the repetition step. I will leave this up to you.

The M step

According to the textbook, in the update for the M step:

$$\mu_j^{new} \propto \sum_i \mathbf{x_i} * W[i,j]$$

Let's deal with the nominator for now. As a review:

$$W = \left[\begin{array}{c|c|c|c} & & & \\ \mathbf{w}_1 & \mathbf{w}_2 & \cdots & \mathbf{w}_K \\ & & & \end{array}\right]$$

Now, let's try to do the $\sum_i \mathbf{x_i} * W[i,j]$ summation using matrix multiplication. The

$$[\mathbf{w}_j^T]_{1\times N} \cdot X_{N\times K} =$$

multiplication gives you a row matrix(with 1 row, and K columns). Let's do this for all clusters:

$$D_{K\times d} = [W^T]_{K\times N} \cdot X_{N\times d}$$

or simply said:

$$D = W^T \cdot X$$

Now, let's take care of the denominator:

$$G[i,j] = \sum_i w_{ij}$$

I'll do it in one step, since we've done this before multiple times

$$G_{K\times d} = [W^T]_{K\times N} \cdot \mathbf{1}_{N\times 1} \cdot \mathbf{1}_{1\times d}$$

Or simply said:

$$G = W^T \cdot \mathbf{1}_{N \times d}$$

Again, $\mathbf{1}_{N \times 1}$ is for the summation part, and $\mathbf{1}_{1 \times d}$ is for the repetition part. Like you did before, it could probably get faster if using proper summation and repetition functions. (no guarantees though)

Therefore, the new μ matrix is

$$\mu^{new} = D \oslash G$$

or simply said

$$\mu^{new} = (W^T \cdot X) \oslash (W^T \cdot \mathbf{1}_{N \times d})$$

The π^{new} is also can be rewritten as:

$$\pi^{new}_{K \times 1} = \frac{[W^T]_{K \times N} \cdot \mathbf{1}_{N \times 1}}{N}$$

or simply said

$$\pi^{new} = \frac{W^T \cdot \mathbf{1}_{N \times 1}}{N}$$

Question: Can we please do these computations in the log space? I'm really having numerical stability issues...

Answer: Sure, let's do that for π^{new} first.

$$\log(\pi^{new}) = \log(W^T \cdot \mathbf{1}_{N \times 1}) - \log(N)$$

Please note that $\log(N)$ is just a scalar. You can easily subtract a vector from a scalar.

In order to compute $\log(W^T \cdot \mathbf{1}_{N \times 1})$, you can easily apply some $log(sum(exp))$ function over each row of $\log(W)$ matrix, and then transpose the result.

Now, let's deal with μ^{new}_j.

$$\mu^{new} = (W^T \cdot X) \oslash (W^T \cdot \mathbf{1}_{N \times d})$$

Multiplying and dividing by some α we get

$$\mu^{new} = (\alpha * (W^T \cdot X)) \oslash (\alpha * (W^T \cdot \mathbf{1}_{N \times d}))$$

Or equivalently

$$\mu^{new} = ((\alpha * W^T) \cdot X)) \oslash (\alpha * W^T \cdot \mathbf{1}_{N \times d})$$

On the other hand:

$$R = \alpha * W^T = \exp\left[\log(\alpha) + \log(W^T)\right]$$

Therefore

- You can compute the R matrix without any numerical stability complications. First add a scalar to the $\log(W^T)$ matrix, so that the R matrix entries get large enough to go through the exponentiation step without numerical stability complications. You can pick any $\log(\alpha)$ value that would solve the numerical issues.

- We already have computed $\log(W^T \cdot \mathbf{1}_{N\times 1})$ using the $log(sum(exp))$ function for the π^{new} update. You can use the same values for the denominator, but keep in mind that the denominator is $\alpha * W^T \cdot \mathbf{1}_{N\times d}$ and you need to multiply $W^T \cdot \mathbf{1}_{N\times d}$ by α.
- Now evaluate the following
$$\mu^{new} = (R \cdot X) \oslash (\alpha * W^T \cdot \mathbf{1}_{N\times d})$$
and you should be able to get rid of the numerical stability issues.