CS 1810 Spring 2025 Section 6: Clustering and Mixture Models

1 Clustering

1.1 Motivation

We now move onto **unsupervised learning**, where the objective is to learn the structure of unlabeled data. In other words, we are looking for groups, or **clusters** among the data. Clustering algorithms are useful not only for finding groups in data, but also to extract features of the data that summarize the most important information about the data in a compressed way.

1.2 Setup

For most clustering algorithms, we need some kind of a metric to specify the notion of "distance" between the data points. If, for example, the points **x** and **x'** live in some Euclidean space $\mathbb{R}^m$, then the natural choice of such metric is the l_2 distance:

$$||\mathbf{x} - \mathbf{x}'||_2 = \sqrt{\sum_{d=1}^{n} (x_d - x'_d)^2}$$

Now that the metric is well-defined, the next thing we need to do is to decide how many groups we want. Sometimes you know the ideal number of groups in advance (*e.g.* clustering the 26 letters in the alphabet). Other times, you need to decide if you'd like a more compressed representation with more information loss by having the number of groups small, or a less compressed representation with less information loss by having the number of groups large.

Suppose our data set is $\{\mathbf{x}^{(n)}\}_{n=1}^{N}$, then our objective is to find the ideal assignment of the data set to the clusters, by assigning to each of the n data points, a binary **responsibility vector** $\mathbf{r}_i$, which is all zeros except one component, corresponding to the assigned cluster.

1.3 K-Means Algorithm

The idea is to represent each cluster by the point in data space that is the average of the data assigned to it. For some choice of K and random initialization of clusters, the K-Means Algorithm (also called Lloyd's algorithm) is:

Repeat until convergence (none of the responsibility vectors change):

1. For each data point, update its responsibility vector by assigning it to the cluster with the closest mean.

2. For each cluster, update the mean $\{\boldsymbol{\mu}_k\}_{k=1}^K$ to be the mean of the data points currently assigned to that cluster.

1.3.1 Derivation

We begin by defining a loss function that the K-Means Algorithm minimizes via coordinate descent:

$$\mathcal{L}(\mathbf{X}, \{\mathbf{r}_n\}_{n=1}^N, \{\boldsymbol{\mu}_k\}_{k=1}^K) = \sum_{n=1}^N \sum_{k=1}^K r_{nk} ||\mathbf{x}^{(n)} - \boldsymbol{\mu}_k||_2^2$$

First, we want to choose r_i that minimizes the loss, holding all else constant. This is when we assign data points to the clusters with means closest to them:

$$r_{ik} = \begin{cases} 1 & \text{if } k = \arg\min_{k'} ||\mathbf{x}^{(n)} - \boldsymbol{\mu}_{k'}||_2 \\ 0 & \text{otherwise} \end{cases}$$

This is the first step of each iteration of the K-means algorithm!

Second, we want to choose μ_k that minimizes the loss, holding all else constant. For a given k, the squared loss is:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\mu}_k) &= \sum_{n=1}^N r_{nk} ||\mathbf{x}^{(n)} - \boldsymbol{\mu}_k||_2^2 \\ &= \sum_{n=1}^N r_{nk} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_k)^T (\mathbf{x}^{(n)} - \boldsymbol{\mu}_k) \end{aligned}$$

Taking the derivative and setting it to zero,

$$\begin{aligned} \frac{\partial \mathcal{L}(\boldsymbol{\mu}_k)}{\partial \boldsymbol{\mu}_k} &= -2 \sum_{n=1}^N r_{nk} (\mathbf{x}^{(n)} - \boldsymbol{\mu}_k) = 0 \\ \boldsymbol{\mu}_k &= \frac{\sum_{n=1}^N r_{nk} \mathbf{x}^{(n)}}{\sum_{n=1}^N r_{nk}} \end{aligned}$$

This is the second step of each iteration of the K-means algorithm!

1.3.2 Number of Clusters

There is not an especially well justified method to choose the number of clusters when using K-means. One approach is to plot K vs the objective criterion, and look for a "knee" or "kink" where progress slows down.

An advanced method is to use the "gap statistic". But this is out of scope for the course.

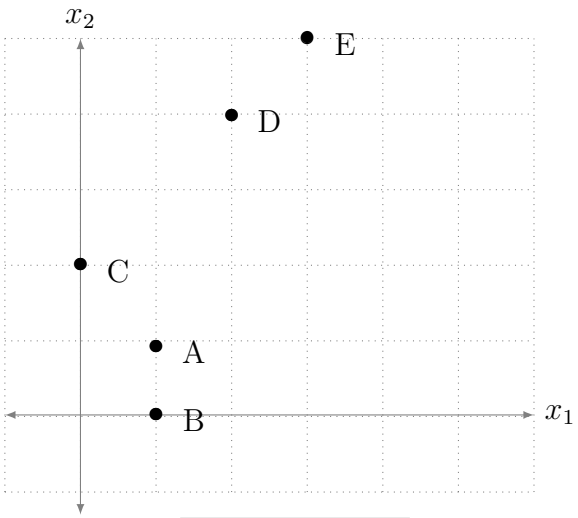
1.3.3 Notes

Lloyd's algorithm finds a locally optimal solution. Finding the globally optimal is NP-hard. A common strategy is to use random restarts. More recently, an algorithm called **K-Means++** has enjoyed popular usage as an alternative to random initialization. This is out of scope for the course, but the basic idea is to randomly select some of the data to be the first cluster centers. This is done by iteratively adding cluster centers, sampling them in proportion to the squared distance of each example from its nearest cluster center. Thus K-Means++ tends to favor points that are distant from the existing centers and produce a more diverse set of centers.

It is generally a good idea to **standardize** the data to account for unsatisfying result due to dimension mismatch. Lastly, when for the metric we are using for the given data set, a "mean" does not make sense, we might instead use a **K-Medoids Algorithm**. This algorithm requires the cluster centers to be a data point in the data set.

1.3.4 Exercise: K-Means (Di Cook)

Use K-means to cluster these examples in $\mathbb{R}^2$, looking for $K = 2$ clusters. Suppose that points A and C are randomly selected as the initial means.



Point	x_1	x_2
A	1	1
B	1	0
C	0	2
D	2	4
E	3	5

1.3.5 Exercise: Convergence of K-Means (Bishop 9.1)

Consider Lloyd's algorithm for finding a K-Means clustering data, i.e., minimizing

$$\mathcal{L}(\{\boldsymbol{r}_n\}_{n=1}^N, \{\boldsymbol{\mu}_k\}_{k=1}^K) = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}||\boldsymbol{x}_i - \boldsymbol{\mu}_k||.$$

Show that as a consequence of there being a finite number of possible assignments for the set of responsibilities r_{ik}, and that for each such assignment there is a unique optimum for the means $\{\boldsymbol{\mu}_k\}_{k=1}^K$, the K-Means algorithm must converge after a finite number of iterations.

1.4 Hierarchical Agglomerative Clustering (HAC)

Hierarchical clustering constructs a tree over the data, where the leaves are individual data items, while the root is a single cluster that contains all of the data. When drawing the dendrogram, for the clustering to be valid, the distances between the two groups being merged should be monotonically increasing. The algorithm is as follows:

1. Start with n clusters, one for each data point.
2. Measure the distance between clusters. This will require an inter-cluster distance measurement that we will define shortly.
3. Merge the two 'closest' clusters together, reducing the number of clusters by 1. Record the distance between these two merged clusters.
4. Repeat step 2 until we're left with only a single cluster.

The main decision in using HAC is what the distance criterion should be between groups. A few examples are given below.

1.4.1 The Min-Linkage Criterion

For two groups indexed by i and i' with n and n' members respectively, the idea is to merge groups based on the shortest distance over all possible pairs:

$$\mathrm{d}_{\min}(\{\mathbf{x}^{(i)}\}_{i=1}^n, \{\mathbf{x}^{(i')}\}_{i'=1}^{n'}) = \min_{i,i'} ||\mathbf{x}^{(i)} - \mathbf{x}^{(i')}||_2.$$

1.4.2 The Max-Linkage Criterion

For two groups indexed by i and i' with n and n' members respectively, the idea is to merge groups based on the largest distance over all possible pairs:

$$\mathrm{d}_{\max}(\{\mathbf{x}^{(i)}\}_{i=1}^n, \{\mathbf{x}^{(i')}\}_{i'=1}^{n'}) = \max_{i,i'} ||\mathbf{x}^{(i)} - \mathbf{x}^{(i')}||_2$$

1.4.3 The Average-Linkage Criterion

For two groups indexed by i and i' with n and n' members respectively, the idea is to average over all possible pairs between the groups:

$$\mathrm{d}_{\mathrm{avg}}(\{\mathbf{x}^{(i)}\}_{i=1}^{n}, \{\mathbf{x}^{(i')}\}_{i'=1}^{n'}) = \frac{1}{nn'} \sum_{i=1}^{n} \sum_{i'=1}^{n'} ||\mathbf{x}^{(i)} - \mathbf{x}^{(i')}||_2$$

1.4.4 The Centroid-Linkage Criterion

For two groups indexed by i and i' with n and n' members respectively, the idea is to look at the difference between the groups' centroids:

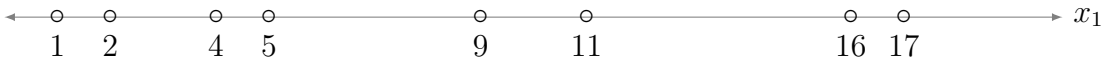
$$\mathrm{d}_{\mathrm{cent}}(\{\mathbf{x}^{(i)}\}_{i=1}^{n}, \{\mathbf{x}^{(i')}\}_{i'=1}^{n'}) = || \left(\frac{1}{n} \sum_{i=1}^{n} \mathbf{x}^{(i)} \right) - \left(\frac{1}{n'} \sum_{i'=1}^{n'} \mathbf{x}^{(i')} \right) ||_2 = ||\bar{\mathbf{x}}_n - \bar{\mathbf{x}}'_{n'}||_2$$

1.4.5 Exercise: K-means and HAC

What are three important differences between K-means and HAC?

1.4.6 Exercise: Min-Linkage and Max-Linkage Criterion

Assume the following examples lie in $\mathbb{R}$. Each example is initially in its own cluster.



{1}{2}{4}{5}{9}{11}{16}{17}

1. Using the Min-Linkage Criterion for the HAC Algorithm, what is the clustering sequence? Draw the dendrogram.
2. Using the Max-Linkage Criterion for the HAC Algorithm, what is the clustering sequence? Draw the dendrogram.

1.4.7 Exercise: Clustering Complexity

What is the "big-O" complexity of HAC? What is the "big-O" complexity of K-means? Compare these.

2 Part II: Mixture Models

2.1 Motivation

A *mixture model* is a type of probabilistic model for unsupervised learning. Suppose you have some observed data $\{\mathbf{x}^{(n)}\}_{n=1}^{N}$. Mixture models are used when you have reason to believe that each individual observation has a discrete *latent variable* $\mathbf{z}^{(n)}$ that determines the data generating process. A latent variable is some piece of data that is unknown, but influences the observed data.

Say there are K possible values for each $\mathbf{z}^{(n)}$, denoted $\{C_k\}_{k=1}^{K}$ where each C_k is a one-hot encoded vector of length K.

Consider the following data-generating process for each data point $\mathbf{x}^{(n)}$:

- Sample latent class $\mathbf{z}^{(n)}$ from $\boldsymbol{\theta}$, the categorical distribution over $\{C_k\}_{k=1}^{K}$ s.t. $p(\mathbf{z} = C_k; \boldsymbol{\theta}) = \theta_k$. Call this sampled latent class C_S.
- Given that $\mathbf{z}^{(n)} = C_S$, sample $\mathbf{x}^{(n)}$ from the distribution

$$p(\mathbf{x}|\mathbf{z} = C_S; \mathbf{w})$$

  This conditional distribution is a modeling assumption (which means we will give it to you in this class), and is specified using unknown parameters $\mathbf{w}$.

  For example, we may assume that $\mathbf{x} \sim p(\mathbf{x}|\mathbf{z} = C_k) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, where $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ are the unknown mean and covariance of the k-th cluster. (See Section 2.4 for more about Gaussian mixture models!)

Example: Say you have a dataset containing weights from a random sample of animals in a pet store. Each $\mathbf{x}^{(n)}$ is the animal's weight. The latent variables $\mathbf{z}^{(n)}$ represent what kind of animal is being weighed, so the possible values $\{C_1, C_2, \ldots, C_K\}$ may represent the categories cat, dog, bird, etc. In your model, you also use the assumption that $p(\mathbf{x}|\mathbf{z} = C_k; \mathbf{w}) \sim N(\mu_k, \sigma_k)$.

Concept Question: In the above example, can you give an intuitive explanation of what vector $\boldsymbol{\theta}$ represents? What does it mean if we define that $p(\mathbf{x}|\mathbf{z} = C_k; \mathbf{w}) \sim N(\mu_k, \sigma_k)$?

2.2 Expectation Maximization

Expectation maximization is a general technique for maximum-likelihood estimation used primarily for models with latent variables. Here we will show how to use EM to train a mixture model, but EM is also used for a variety of other models!

Consider a generative mixture model consisting of latent variables $\boldsymbol{z}^{(n)}$ drawn from a distribution $p(\boldsymbol{z}^{(n)}; \boldsymbol{\theta})$ and observed data $\boldsymbol{X}$, such that we draw each $\boldsymbol{x}^{(n)}$ from a distribution $p(\boldsymbol{x}^{(n)}|\boldsymbol{z}^{(n)}; \mathbf{w})$.

We have 2 goals:

1. To compute the MLE for $\mathbf{w}$ and $\boldsymbol{\theta}$, i.e. the values of $\mathbf{w}$, $\boldsymbol{\theta}$ that maximize $p(\boldsymbol{X}; \mathbf{w}, \boldsymbol{\theta})$.

2. To estimate the latent variable $\mathbf{z}^{(n)}$ corresponding to a particular $\mathbf{x}^{(n)}$, which is captured by the posterior distribution $p(\mathbf{z}^{(n)}|\mathbf{x}^{(n)}; \mathbf{w}, \boldsymbol{\theta})$.

Goal 2 is easy once we have an estimate of the MLE for $\mathbf{w}, \boldsymbol{\theta}$, because we can apply Bayes' rule:

$$p(\mathbf{z}|\mathbf{x}; \mathbf{w}, \boldsymbol{\theta}) \propto p(\mathbf{x}|\mathbf{z}; \mathbf{w}, \boldsymbol{\theta}) p(\mathbf{z}; \mathbf{w}, \boldsymbol{\theta})$$

$$p(\mathbf{z}|\mathbf{x}; \mathbf{w}, \boldsymbol{\theta}) \propto p(\mathbf{x}|\mathbf{z}; \mathbf{w}) p(\mathbf{z}; \boldsymbol{\theta}) \tag{1}$$

2.2.1 Why EM?

By LOTP, the log-likelihood of the data $\mathbf{X}$ can be derived as such:

$$p(\mathbf{X}; \mathbf{w}, \boldsymbol{\theta}) = \prod_{n=1}^{N} \mathbb{E}_{\boldsymbol{z}^{(n)} \sim p(\boldsymbol{z}^{(n)}; \boldsymbol{\theta})} \left[p(\mathbf{x}^{(n)}|\mathbf{z}^{(n)}; \mathbf{w}) \right] \tag{2}$$

$$\log p(\mathbf{X}; \mathbf{w}, \boldsymbol{\theta}) = \sum_{n=1}^{N} \log \mathbb{E}_{\boldsymbol{z}^{(n)} \sim p(\boldsymbol{z}^{(n)}; \boldsymbol{\theta})} \left[p(\mathbf{x}^{(n)}|\mathbf{z}^{(n)}; \mathbf{w}) \right] \tag{3}$$

There is no closed form for the MLE of this log-likelihood due to two main obstacles:

1. First, the expectation being inside the log poses a computational barrier (you can see why this is true when the expectation is a sum in the discrete case).

2. Second, the expectation itself depends on $\boldsymbol{\theta}$, in the sense that the random quantity $\boldsymbol{z}^{(n)}$ is drawn from a distribution parameterized by $\boldsymbol{\theta}$. We are looking to compute a gradient with respect to $\boldsymbol{\theta}$, but in such a case, gradients do not commute with expectations. Specifically, it is not generally true that

   $$\nabla_{\boldsymbol{\theta}} \mathbb{E}_{x \sim p(x; \boldsymbol{\theta})} \left[f(x; \boldsymbol{\theta}) \right] = \mathbb{E}_{x \sim p(x; \boldsymbol{\theta})} \left[\nabla_{\boldsymbol{\theta}} f(x; \boldsymbol{\theta}) \right]$$

   This stops us from being able to optimize the log-likelihood. However, if the distribution of x in the above example did not depend on $\boldsymbol{\theta}$, the equality would be true.

2.2.2 ELBO: The Evidence Lower Bound

We now derive the **ELBO**, an alternative to the log-likelihood that we maximize in our final training algorithm. The idea is to address obstacle 1 through using Jensen's inequality to swap the order of the log and the expectation, and to address obstacle 2 through using a proxy distribution $q^{(n)}$ in place of $p(\boldsymbol{z}^{(n)}; \boldsymbol{\theta})$ in the expectation. Mathematically, we have

$$\begin{aligned}
\log p(\mathbf{X}; \mathbf{w}, \boldsymbol{\theta}) &= \sum_{n=1}^{N} \log \mathbb{E}_{\boldsymbol{z}^{(n)} \sim p(\boldsymbol{z}^{(n)}; \boldsymbol{\theta})} \left[p(\mathbf{x}^{(n)} | \mathbf{z}^{(n)}; \mathbf{w}) \right] \\
&= \sum_{n=1}^{N} \log \mathbb{E}_{\boldsymbol{z}^{(n)} \sim p(\boldsymbol{z}^{(n)}; \boldsymbol{\theta})} \left[p(\mathbf{x}^{(n)} | \mathbf{z}^{(n)}; \mathbf{w}) \cdot \frac{q^{(n)}(\boldsymbol{z}^{(n)})}{q^{(n)}(\boldsymbol{z}^{(n)})} \right] \\
&= \sum_{n=1}^{N} \log \mathbb{E}_{\boldsymbol{z}^{(n)} \sim q^{(n)}(\boldsymbol{z}^{(n)})} \left[\frac{p(\mathbf{x}^{(n)} | \mathbf{z}^{(n)}; \mathbf{w}) p(\boldsymbol{z}^{(n)}; \boldsymbol{\theta})}{q^{(n)}(\boldsymbol{z}^{(n)})} \right] \\
&= \sum_{n=1}^{N} \log \mathbb{E}_{\boldsymbol{z}^{(n)} \sim q^{(n)}(\boldsymbol{z}^{(n)})} \left[\frac{p(\mathbf{x}^{(n)}, \mathbf{z}^{(n)}; \mathbf{w}, \boldsymbol{\theta})}{q^{(n)}(\boldsymbol{z}^{(n)})} \right] \\
&\geq \sum_{n=1}^{N} \mathbb{E}_{\boldsymbol{z}^{(n)} \sim q^{(n)}(\boldsymbol{z}^{(n)})} \left[\log \left(\frac{p(\mathbf{x}^{(n)}, \mathbf{z}^{(n)}; \mathbf{w}, \boldsymbol{\theta})}{q^{(n)}(\boldsymbol{z}^{(n)})} \right) \right] \\
&= \sum_{n=1}^{N} \mathbb{E}_{\boldsymbol{z}^{(n)} \sim q^{(n)}(\boldsymbol{z}^{(n)})} \left[\log p(\mathbf{x}^{(n)}, \mathbf{z}^{(n)}; \mathbf{w}, \boldsymbol{\theta}) - \log q^{(n)}(\boldsymbol{z}^{(n)}) \right]
\end{aligned}$$

This final term is the ELBO. The key derivations above are the third line, in which we switch what we take the expectation over, and the fifth line, in which we apply Jensen's inequality. Now we are ready to describe expectation maximization.

2.2.3 The EM Algorithm

Since finding the MLE directly is difficult, we will use **expectation maximization**, which uses $\boldsymbol{q}$ and the ELBO instead of the log-likelihood. EM is an approximate iterative approach that continually updates $\boldsymbol{q}, \boldsymbol{w}, \boldsymbol{\theta}$ with the following steps:

1. Initialize $\mathbf{w}^{(0)}, \boldsymbol{\theta}^{(0)}$ randomly.
2. *(E-step)* Use the current parameters $\boldsymbol{w}^{(t)}, \boldsymbol{\theta}^{(t)}$ to update the distribution $q^{(n)}$ for each example:

$$q^{(n)}(\boldsymbol{z}^{(n)}) = \mathbb{E}_{p(\boldsymbol{z}^{(n)}|\boldsymbol{x}^{(n)};\boldsymbol{w}^{(t)},\boldsymbol{\theta}^{(t)})}[\boldsymbol{z}^{(n)}] = \begin{bmatrix} p(\mathbf{z}^{(n)} = C_1|\mathbf{x}^{(n)};\mathbf{w}^{(t)},\boldsymbol{\theta}^{(t)}) \\ \vdots \\ p(\mathbf{z}^{(n)} = C_k|\mathbf{x}^{(n)};\mathbf{w}^{(t)},\boldsymbol{\theta}^{(t)}) \end{bmatrix}$$
$$\propto \begin{bmatrix} p(\mathbf{x}^{(n)}|\mathbf{z}^{(n)} = C_1;\mathbf{w}^{(t)})p(\mathbf{z}^{(n)} = C_1;\boldsymbol{\theta}^{(t)}) \\ \vdots \\ p(\mathbf{x}^{(n)}|\mathbf{z}^{(n)} = C_k;\mathbf{w}^{(t)})p(\mathbf{z}^{(n)} = C_k;\boldsymbol{\theta}^{(t)}) \end{bmatrix}$$

3. *(M-step)* Update parameters: Choose the value of $\mathbf{w}^{(t+1)}, \boldsymbol{\theta}^{(t+1)}$ that maximizes the ELBO, where the expectation is over the current distributions $\boldsymbol{q}^{(n)}$ that we just updated above:

$$\begin{aligned} \mathbf{w}^{(t+1)}, \boldsymbol{\theta}^{(t+1)} &= \arg\max_{\boldsymbol{w},\boldsymbol{\theta}} \sum_{n=1}^{N} \mathbb{E}_{\boldsymbol{z}^{(n)}\sim q^{(n)}(\boldsymbol{z}^{(n)})} \left[\log p(\mathbf{x}^{(n)}, \mathbf{z}^{(n)}; \mathbf{w}, \boldsymbol{\theta}) - \log q^{(n)}(\boldsymbol{z}^{(n)})\right] \\ &= \arg\max_{\boldsymbol{w},\boldsymbol{\theta}} \sum_{n=1}^{N} \mathbb{E}_{\boldsymbol{z}^{(n)}\sim q^{(n)}(\boldsymbol{z}^{(n)})} \left[\log p(\mathbf{x}^{(n)}, \mathbf{z}^{(n)}; \mathbf{w}, \boldsymbol{\theta})\right] \end{aligned}$$

   The reason we can drop the second term in the ELBO is that it is free of the true parameters $\boldsymbol{w}, \boldsymbol{\theta}$. Recall that this is because we designed $q^{(n)}$ to not depend on these true parameters and instead it only uses the current estimates $\boldsymbol{w}^{(t)}, \boldsymbol{\theta}^{(t)}$, which are different! You can think of the remaining expression in the second line as an "expected complete data log-likelihood."

4. Go back to step 2 until the log-likelihood estimate in step 3 converges.

2.2.4 Example: Gaussian Mixture Model

Recall from lecture the following setup:

We have data $\mathbf{x}^{(n)} \in \mathbb{R}^D$ and a latent variable $\mathbf{z}^{(n)}$ (corresponding to the cluster that the point is drawn from) such that $\mathbf{x} \sim p(\mathbf{x}|\mathbf{z} = C_k) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, where $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ are the mean and covariance of the k-th cluster. The choice of cluster is drawn from a categorical distribution with probabilities $\boldsymbol{\pi} \in [0, 1]^K$. We are able to observe the data $\mathbf{x}^{(n)}$ and want to find the cluster centers and their covariances.

The steps of EM inference applied to this problem are:

1. Randomly initialize $\boldsymbol{\pi}, \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^K$.

2. Next, calculate the new distribution of each $\mathbf{z}^{(n)}$:

$$q_{n,k} = p(z^{(n)} = C_k|\mathbf{x}^{(n)}) \propto \pi_k \mathcal{N}(\mathbf{x}^{(n)}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

   This is our new estimate of the distribution of $\mathbf{z}^{(n)}$ given the data and our estimate for $\boldsymbol{\pi}, \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_k$.

3. Find the expected complete data log-likelihood (first term of ELBO):

$$\mathbb{E}_{\boldsymbol{z}^{(n)}\sim q^{(n)}(\boldsymbol{z}^{(n)})}\left[\log p(\mathbf{x}^{(n)}, \mathbf{z}^{(n)}; \mathbf{w}, \boldsymbol{\theta})\right] = \sum_{n=1}^{N}\sum_{k=1}^{K} q_{n,k}\ln\pi_k + q_{n,k}\ln\mathcal{N}(\mathbf{x}^{(n)}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

and then optimize it for each of the parameters $\boldsymbol{\pi}, \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\}_{k=1}^{K}$. However, we need to be careful to remember constraints: since $\sum_k \pi_k = 1$, we must use Lagrange multipliers to optimize the parameters. We get the following update equations:

$$\begin{aligned}
\pi_k^{(t+1)} &= \frac{\sum_{n=1}^{N} q_{n,k}}{N} \\
\boldsymbol{\mu}_k^{(t+1)} &= \frac{\sum_{n=1}^{N} q_{n,k}\mathbf{x}^{(n)}}{\sum_{n=1}^{N} q_{n,k}} \\
\boldsymbol{\Sigma}_k^{(t+1)} &= \frac{\sum_{n=1}^{N} q_{n,k}(\mathbf{x}^{(n)} - \boldsymbol{\mu}_k^{(t+1)})(\mathbf{x}^{(n)} - \boldsymbol{\mu}_k^{(t+1)})^\top}{\sum_{n=1}^{N} q_{n,k}}
\end{aligned}$$