In the problem of **clustering**, we are given a dataset comprised only of input features without labels. We wish to assign to each data point a discrete label indicating which "cluster" it belongs to, in such a way that the resulting cluster assignment "fits" the data. We are given flexibility to choose our notion of goodness of fit for cluster assignments.

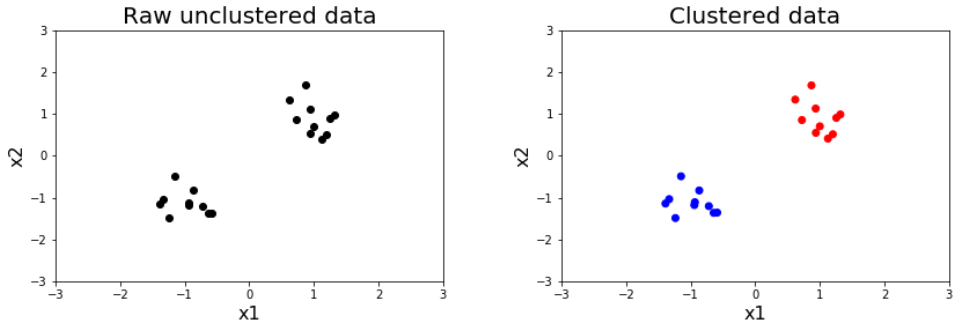

Figure 1: Left: unclustered raw data; Right: clustered data

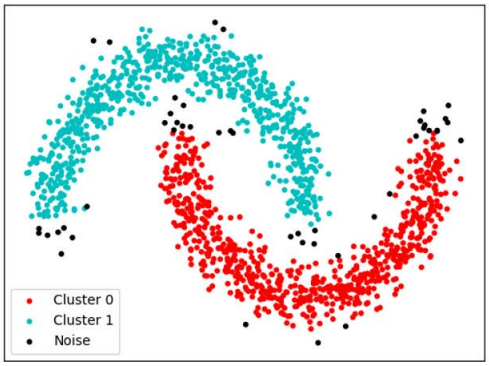

Figure 2: A nonspherical clustering assignment. Possible outliers are shown in black.[1]

In our discussion of LDA and QDA, we assumed that we had data which was conditionally Gaussian given a discrete class label. When we observed a data point, we observed both its input features and its class label. These are **supervised** learning methods, which deal with prediction of observed outputs from observed inputs. Clustering is an example of **unsupervised learning**, where we are not given labels and desire to infer something about the underlying structure of the data. Another example of unsupervised learning is dimensionality reduction, where we desire to learn important features from the data.

Clustering is most often used in exploratory data visualization, as it allows us to see the different

[1] https://www.imperva.com/blog/2017/07/clustering-and-dimensionality-reduction-understanding-the-magic-behind-machine-learning/

groups of similar data points within the data. Combined with domain knowledge, these clusters can have a physical interpretation - for example, different clusters can represent different species of plant in the biological setting, or types of consumers in a business setting. If desired, these clusters can be used as pre-processing to make the data more compact. Clustering is also used for outlier detection, as in Figure 2: data points that do not seem to belong in their assigned cluster may be flagged as outliers.

In order to create an algorithm for clustering, we first must determine what makes a good clustering assignment. Here are some possible desired properties:

1. High intra-cluster similarity - points within a given cluster are very similar.
2. Low inter-cluster similarity - points in different clusters are not very similar.

Of course, this depends on our notion of similarity. For now, we will say that points in $\mathbb{R}^d$ are similar if their L^2 distance is small, and dissimilar otherwise. A generalization of this notion is provided in the appendix.

1 K-means Clustering

Let X denote the set of N data points $\mathbf{x}_i \in \mathbb{R}^d$. A *cluster assignment* is a partition $C_1, ..., C_K \subseteq X$ such that the sets C_k are disjoint and $X = C_1 \cup \cdots \cup C_K$. A data point $\mathbf{x} \in X$ is said to belong to cluster k if it is in C_k.

One approach to the clustering problem is to represent each cluster C_k by a single point $\mathbf{c}_k \in \mathbb{R}^d$ in the input space - this is called the **centroid** approach. K-means is an example of centroid-based clustering where we choose centroids and a cluster assignment such that the total distance of each point to its assigned centroid is minimized. In this regard, K-means optimizes for high intra-cluster similarity, but the clusters do not necessarily need to be far apart, so we may also have high inter-cluster similarity.

Formally, K-means solves the following problem:

$$\underset{\{C_k\}_{k=1}^K, \{\mathbf{c}_k\}_{k=1}^K : X = C_1 \cup \cdots \cup C_K}{\arg\min} \sum_{k=1}^{K} \sum_{\mathbf{x} \in C_k} \|\mathbf{x} - \mathbf{c}_k\|^2$$

It has been shown that this problem is NP hard, so solving it exactly is intractable. However, we can come up with a simple algorithm to compute a candidate solution. If we knew the cluster assignment $C_1, ..., C_K$, then we would only need to determine the centroid locations. Since the choice of centroid location $\mathbf{c}_i$ does not affect the distances of points in C_j to $\mathbf{c}_j$ for $i \neq j$, we can consider each cluster separately and choose the centroid that minimizes the sum of squared distances to points in that cluster. The centroid we compute, $\hat{\mathbf{c}}_k$, is

$$\hat{\mathbf{c}}_k = \underset{\mathbf{c}_k}{\arg\min} \sum_{\mathbf{x} \in C_k} \|\mathbf{x} - c_k\|^2$$

But this is simply the mean of the data in C_k, that is,

$$\hat{\mathbf{c}}_k = \frac{1}{|C_k|} \sum_{\mathbf{x} \in C_k} \mathbf{x}$$

Similarly, if we knew the centroids $\mathbf{c}_k$, in order to choose the cluster assignment $C_1, ..., C_K$ that minimizes the sum of squared distances to the centroids, we simply assign each data point $\mathbf{x}$ to the cluster represented by its closest centroid, that is, we assign $\mathbf{x}$ to

$$\arg\min_k \ \|\mathbf{x} - \mathbf{c}_k\|^2$$

Now we can perform alternating minimization - on each iteration of our algorithm, we update the clusters using the current centroids, and then update the centroids using the new clusters. This algorithm is sometimes called **Lloyd's Algorithm**.

Algorithm 1: K-means Algorithm

Initialize $\mathbf{c}_k$, $k = 1, ..., K$

while *K-means objective has not converged* **do**

- Update partition $C_1 \cup \cdots \cup C_K$ given the $\mathbf{c}_k$ by assigning each $x \in X$ to the cluster represented by its nearest centroid
- Update centroids $\mathbf{c}_k$ given $C_1 \cup \cdots \cup C_K$ as $\mathbf{c}_k = \frac{1}{|C_k|} \sum_{\mathbf{x} \in C_k} \mathbf{x}$

This algorithm will always converge to some value. To show this, note the following facts:

1. There are only finitely many (say, M) possible partition/centroid pairs that can be produced by the algorithm. This is true since each partition chosen at some iteration in the algorithm completely determines the subsequent centroid assignment in that iteration.
2. Each update of the cluster assignment and centroids does not increase the value of the objective. This is true since each of these updates is a minimization of the objective which we solve exactly.

If the value of the objective has not converged after M iterations, then we have cycled through all the possible partition/centroid pairs attainable by the algorithm. On the next iteration, we would obtain a partition and centroid assignment that we have already seen, say on iteration $t \leq M$. But this means that the value of the objective at time $M + 1$ is the same as at time t, and because the value of the objective function never increases throughout the algorithm, the value is the same as at time M, so we have converged.

In practice, it is common to run the K-means algorithm multiple times with different initialization points, and the cluster corresponding to the minimum objective value is chosen. There are also ways to choose a smarter initialization than a random seed, which can improve the quality of the local optimum found by the algorithm.[2] It should be emphasized that no efficient algorithm for

[2] For example, K-means++.

solving the K-means optimization is guaranteed to give a good cluster assignment, as the problem is NP hard and there are local optima.

Choosing the number of clusters k is similar to choosing the number of principal components for PCA - we can compute the value of the objective for multiple values of k and find the "elbow" in the curve.

We have noted that the main algorithm for solving K-means does not have to produce a good solution. Let us step back and consider some shortcomings of the K-means objective function itself:

1. There is no likelihood attached to K-means, which makes it harder to understand what assumptions we are making on the data.
2. Each feature is treated equally, so the clusters produced by K-means will look spherical. We can also infer this by looking at the sum of squares in the objective function, which we have seen to be related to spherical Gaussians.
3. Each cluster assignment in the optimization is a **hard assignment** - each point belongs in exactly one cluster. A **soft assignment** would assign each point to a distribution over the clusters, which can encode not only which cluster a point belongs to, but also how far it was from the other clusters.

1.1 Soft K-means

We can introduce soft assignments to our algorithm easily using the familiar softmax function. Recall that if $\mathbf{z} \in \mathbb{R}^d$, then the softmax function σ is defined as

$$\sigma(\mathbf{z})_j = \frac{e^{\mathbf{z}_j}}{\sum_{k=1}^{d} e^{\mathbf{z}_k}}$$

To compute the cluster assignment of a data point $\mathbf{x}_i$ in K-means, we computed

$$\arg\min_k \|\mathbf{x}_i - \mathbf{c}_k\|^2 = \arg\max_k -\|\mathbf{x}_i - \mathbf{c}_k\|^2$$

In soft K-means, we instead compute a soft assignment $r_i(k), k = 1, ..., K$ where $\sum_k r_i(k) = 1$ as the softmax of the vector of $\mathbf{z} := -\beta\|\mathbf{x}_i - \mathbf{c}_k\|^2$, $k = 1, ..., K$:

$$r_i(k) = \sigma(\mathbf{z})_k$$

Here, β is a tunable parameter indicating the level of "softness" desired.

Once we have computed the soft assignments, we may use them to determine the centroids by using a weighted average. Before, we defined the new centroids as

$$\hat{\mathbf{c}}_k = \arg\min_{\mathbf{c}_k} \sum_{\mathbf{x} \in C_k} \|\mathbf{x} - \mathbf{c}_k\|^2$$

Now, we apply the $r_i(k)$ as weights for the minimization:

$$\hat{\mathbf{c}}_k = \arg\min_{\mathbf{c}_k} \sum_{i=1}^{N} r_i(k)\|\mathbf{x}_i - \mathbf{c}_k\|^2 = \frac{\sum_{i=1}^{N} r_i(k)\mathbf{x}_i}{\sum_{i=1}^{N} r_i(k)} \tag{1}$$

This is now a weighted average of the $\mathbf{x}_i$ - the weights reflect how much we believe each data point belongs to a particular cluster, and because we are using this information, our algorithm should not jump around between clusters, resulting in better convergence speed.

There are still a few issues with soft K-means. One is the choice of β - it is not so clear how to set this hyperparameter. Another issue is that our clusters are still spherical, since we are still weighting all features the same (note that we have weighted each *data point* differently with soft K-means). To solve these issues, we will use a fully probabilistic model.

2 Mixture of Gaussians

Suppose $\boldsymbol{\mu}_k \in \mathbb{R}^d, \boldsymbol{\Sigma}_k \in \mathbb{R}^{d \times d}$ are fixed parameters for $k = 1, ..., K$. Consider the following experiment: draw a value z from some distribution on the set of indices $\{1, ..., K\}$, and then draw $\mathbf{x} \in \mathbb{R}^d$ from the Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z)$. We can interpret $\mathbf{x}$ as belonging to cluster z. This model is called **Mixture of Gaussians (MoG)**, also known as a **Gaussian mixture model**.

If we have fit a MoG model to data (ie. we have determined values for $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$, and the prior on z), then to perform clustering, we can use Bayes' rule to determine the posterior $P(z = k|\mathbf{x})$ and assign $\mathbf{x}$ to the cluster k that maximizes this quantity. In fact, this is exactly our decision rule with QDA using a prior - the difference is that QDA, a supervised method, is given labels to fit the mixture model, while in the unsupervised clustering setting we must fit the mixture model without the aid of labels. When $\boldsymbol{\Sigma}_k$ are not multiples of the identity, we can obtain non-spherical clusters, which was not possible with K-means.

MoG is an example of a **latent variable model**. A latent variable model is a probabilistic model in which some variables can be directly observed or measured, while other latent (hidden) variables cannot be observed directly; rather, we observe them indirectly through their influence on the observed variables. When we try to fit a MoG model to data, we only observe the data $\mathbf{x}_i$, which we presume to have been generated based on the latent variable z_i, the cluster assignment. Latent variable models are modular and can be used to model complex dependencies in a probabilistic model - however, the added flexibility can lead to difficulty in learning its parameters.

To illustrate this, we will examine the likelihood function for MoG. Suppose $\mathbf{x}_i$ has distribution $p(\mathbf{x}_i; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ is a set of all $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, p(z_i = k)$. The likelihood for the single data point $\mathbf{x}_i$ is

$$\begin{aligned}
\mathcal{L}_i(\boldsymbol{\theta}; \mathbf{x}_i) &= p(\mathbf{x}_i; \boldsymbol{\theta}) \\
&= \sum_{k=1}^{K} p(\mathbf{x}_i, z_i = k; \boldsymbol{\theta}) \\
&= \sum_{k=1}^{K} p(\mathbf{x}_i | z_i = k; \boldsymbol{\theta}) p(z_i = k; \boldsymbol{\theta})
\end{aligned}$$

Over all data points, the likelihood is

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}; \mathbf{x}) &= \prod_{i=1}^{N} \mathcal{L}_i(\boldsymbol{\theta}; \mathbf{x}_i) \\ &= \prod_{i=1}^{N} \sum_{k=1}^{K} p(\mathbf{x}_i | z_i = k; \boldsymbol{\theta}) p(z_i = k; \boldsymbol{\theta})\end{aligned}$$

Hence the log likelihood $\ell(\boldsymbol{\theta}; \mathbf{x})$ is given by

$$\ell(\boldsymbol{\theta}; \mathbf{x}) = \sum_{i=1}^{N} \log \left(\sum_{k=1}^{K} p(\mathbf{x}_i | z_i = k; \boldsymbol{\theta}) p(z_i = k; \boldsymbol{\theta}) \right) \tag{2}$$

When we perform QDA, we know the z_i are known, deterministic quantities and thus the likelihood (2) reduces to

$$\ell(\boldsymbol{\theta}; \mathbf{X}) = \sum_{i=1}^{N} \log p(\mathbf{x}_i | z_i; \boldsymbol{\theta})$$

Maximizing this is equivalent to fitting the individual class-conditional Gaussians via maximum likelihood, which is consistent with how we have described QDA in the past. When we fit the MoG without knowledge of the latent variables, the parameters $\boldsymbol{\theta}$ in (2) are now coupled together inside the log, which complicates the likelihood. While it is still possible to find the MLE by working out the gradient and using our descent methods, there is an alternative approach called Expectation-Maximization (EM), which takes advantage of the latent variable structure.

3 Expectation Maximization (EM) Algorithm

The **Expectation-Maximization (EM) Algorithm** is used to compute the MLE for latent variable models, such as MoG. First recall that soft K-means consisted of the following two alternating steps:

1. For each data point, compute a soft assignment $r_i(k)$ to the clusters - that is, a probability distribution over clusters. The soft assignment is obtained by using a softmax.
2. Update the centroids in an optimal way given the soft assignments computed in the first step. The resulting updates are a weighted average of the data points.

One could rephrase these updates as:

1. Soft imputation of the data - fill in the missing data ("impute") with a probability distribution over all its possible values ("soft").
2. Parameter updates given the imputed data

The EM algorithm alternates between these two steps in the same way as soft K-means, but the updates for each step are performed in a principled way to maximize certain "components" of the log likelihood (we will make this precise later). We will derive the following updates for EM when computing the MLE for MoG:

1. Using the current parameter estimates, estimate $p(z_i|\mathbf{x}_i;\boldsymbol{\theta})$. That is, perform soft imputation of the latent cluster variable.
2. Estimate the parameters via MLE, using the estimates of $p(z_i|\mathbf{x}_i;\boldsymbol{\theta})$ to make the computation tractable.

Both soft K-means and EM estimate $p(z_i|\mathbf{x}_i;\boldsymbol{\theta})$, though EM will do so in a more principled way. In the second alternating step, we will see that the EM update of the mean is exactly the same as the soft K-means centroid update. However, EM will also update the covariance estimates, which captures ellipsoidal structure in the data.

To derive the EM algorithm, it will be helpful to introduce the notion of the **complete log likelihood**, which we define as

$$L_c(\mathbf{x}_i, z_i;\boldsymbol{\theta}) := \log p(\mathbf{x}_i, z_i;\boldsymbol{\theta})$$

If we assumed z_i to be known, this would be the log likelihood of the data. In practice, we do not know the values of z_i, but if we are given a distribution $q(z_i|\mathbf{x}_i)$ over the latents z_i, we can marginalize over the possible values of z_i by taking the expectation

$$\mathbb{E}_q(L_c(\mathbf{x}_i, z_i;\boldsymbol{\theta})) = \sum_{k=1}^{K} L_c(\mathbf{x}_i, z_i = k;\boldsymbol{\theta}) q(z_i = k|\mathbf{x}_i)$$

We call this the **expected complete log likelihood**. The distribution q is an estimate of the true conditional distribution $p(z_i|\mathbf{x}_i;\boldsymbol{\theta})$; in the EM algorithm, we will alternate between updating our q distribution to better estimate $p(z_i\mathbf{x}_i;\boldsymbol{\theta})$ and maximizing the expected complete log likelihood in place of the true likelihood function.

We are now in a position to derive the algorithm. We will need a well-known result called **Jensen's Inequality**:

Theorem 1. *If X is a random variable and f is convex, then $f(\mathbb{E}(X)) \leq \mathbb{E}(f(X))$.*

If f is concave, then using the fact $-f$ is convex immediately yields the conclusion $\mathbb{E}(f(X)) \leq f(\mathbb{E}(X))$. In particular, since $\log$ is concave, we have $\mathbb{E}(\log(X)) \leq \log\big(\mathbb{E}(X)\big)$.

Now we can derive the EM algorithm. Suppose $\mathbf{x}_i$ are random variables depending on z_i, and $\boldsymbol{\theta}$ are the parameters of interest. Given any conditional distribution over the latents $q(z_i = k|\mathbf{x}_i)$, the log likelihood for the i-th data point is

$$\begin{aligned}\ell_i(\boldsymbol{\theta};\mathbf{x}_i) &= \log p(\mathbf{x}_i;\boldsymbol{\theta}) \\ &= \log \sum_{k=1}^{K} p(\mathbf{x}_i, z_i = k;\boldsymbol{\theta})\end{aligned}$$

$$
\begin{aligned}
&= \log \sum_{k=1}^{K} \frac{q(z_i = k|\mathbf{x}_i) p(\mathbf{x}_i, z_i = k; \boldsymbol{\theta})}{q(z_i = k|\mathbf{x}_i)} \\
&= \log \mathbb{E}_q \left[\frac{p(\mathbf{x}_i, z_i; \boldsymbol{\theta})}{q(z_i|\mathbf{x}_i)} \right] \\
&\geq \mathbb{E}_q \left[\log \frac{p(\mathbf{x}_i, z_i; \boldsymbol{\theta})}{q(z_i|\mathbf{x}_i)} \right] \quad \text{(Jensen)} \\
&= \sum_{k=1}^{K} q(z_i = k|\mathbf{x}_i) \log [\frac{p(\mathbf{x}_i, z_i = k; \boldsymbol{\theta})}{q(z_i = k|\mathbf{x}_i)}] \\
&= -\sum_{k=1}^{K} q(z_i = k|\mathbf{x}_i) \log [q(z_i = k|\mathbf{x}_i)] + \sum_{k=1}^{K} q(z_i = k|\mathbf{x}_i) \log p(\mathbf{x}_i, z_i = k; \boldsymbol{\theta}) \\
&=: F_i(q, \boldsymbol{\theta})
\end{aligned}
$$

We will define

$$H(q(z_i|\mathbf{x}_i)) := -\sum_{k=1}^{K} q(z_i = k|\mathbf{x}_i) \log [q(z_i = k|\mathbf{x}_i)]$$

and

$$L_c(\mathbf{x}_i, z_i; \boldsymbol{\theta}) := \log p(\mathbf{x}_i, z_i; \boldsymbol{\theta})$$

so that the above lower bound can be written as

$$F_i(q, \boldsymbol{\theta}) = H(q(z_i|\mathbf{x}_i)) + \mathbb{E}_q(L_c(\mathbf{x}_i, z_i; \boldsymbol{\theta}))$$

The first term $H(q(z_i|\mathbf{x}_i))$ has an information-theoretic interpretation - it is the **entropy** of the distribution $q(z_i|\mathbf{x}_i)$, a non-negative quantity that measures the amount of disorder encoded in the distribution. As mentioned earlier, the term $L_c(\mathbf{x}_i, z_i; \boldsymbol{\theta})$ is called the **complete log likelihood** of $\mathbf{x}_i$ - it will be easier to optimize $\mathbb{E}_q(L_c(\mathbf{x}_i, z_i; \boldsymbol{\theta}))$ than the original log likelihood, since we will not need to deal with the "marginalization problem" in the original log likelihood from Equation 2.

Since the log likelihood of the full data is the sum of the individual log likelihoods, we can take sums and find

$$\ell(\boldsymbol{\theta}; \mathbf{X}) \geq \sum_{i=1}^{N} F_i(q, \boldsymbol{\theta}) = H(q(\mathbf{z}|\mathbf{X})) + \mathbb{E}_q(L_c(\mathbf{X}, \mathbf{z}; \boldsymbol{\theta})) =: F(q, \boldsymbol{\theta}) \tag{3}$$

Here, $\mathbf{X}$ denotes the full dataset and $\mathbf{z}$ denotes the length N vector of latent variables. It is easy to check that if $q(z_i|\mathbf{x}_i) = p(z_i|\mathbf{x}_i; \boldsymbol{\theta})$ for all i, then the inequality (3) is tight (set $q(z_i|\mathbf{x}_i) = p(\mathbf{x}_i, z_i; \theta)$ in the application of Jensen's inequality and observe both sides of the inequality are equal). Thus it makes sense to perform an alternating maximization scheme, where we iteratively update $q(z_i|\mathbf{x}_i)$ to $p(z_i|\mathbf{x}_i; \boldsymbol{\theta})$ to make the inequality tight and then maximize over $\boldsymbol{\theta}$. Formally, the algorithm is as follows:

1. Initialize $\boldsymbol{\theta}^0$

2. Expectation (E) step (soft imputation): set $q^{t+1} = \arg\max_q F(q, \boldsymbol{\theta}^t)$, that is,

$$q^{t+1}(z_i = k|\mathbf{x}_i) := p(z_i = k|\mathbf{x}_i; \boldsymbol{\theta}^t)$$

   This value of q is used to compute $\mathbb{E}_q(L_c(\mathbf{X}, \mathbf{z}; \boldsymbol{\theta}^t))$.

3. Maximization (M) step (parameter estimation): set

$$\boldsymbol{\theta}^{t+1} = \arg\max_{\boldsymbol{\theta}} F(q^{t+1}, \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{q^{t+1}}(L_c(\mathbf{X}, \mathbf{z}; \boldsymbol{\theta}))$$

4. Repeat steps 2, 3 until convergence

A few remarks: when we maximize over q in the E step, there are nK values to be updated - one value of $q(z_i = k|\mathbf{x}_i)$ for every data index i and latent index k. After the E step, q^{t+1} is fixed and does not depend on $\boldsymbol{\theta}$, so the entropy term does not depend on $\boldsymbol{\theta}$ and maximizing $F(q^{t+1}, \boldsymbol{\theta})$ in the subsequent M step amounts to maximizing the expected complete log likelihood. The E step is what we have previously described as soft imputation of the latents: we fill in values for the hidden variables z_i by determining a conditional distribution $q(z_i|\mathbf{x}_i)$. The M step assumes the E step has done a reasonable job at imputing the data and uses this additional information to maximize the likelihood. Observe the connections between K-means, soft K-means, and EM - all perform alternating steps of data imputation and subsequent parameter optimization given the imputed data. In the data imputation step for K-means, each data point is given a hard assignment to a latent variable value, while in soft K-means and EM, each data point gets assigned a *distribution* over the latent variables.

From our derivation of EM, we can see that the value of the likelihood never decreases during the execution of the algorithm. It turns out that EM will converge to a parameter estimate with zero gradient, but will not necessarily find the global optimum. When the clusters are sufficiently separated, EM can exhibit Newtonian (second-order) convergence speed - however, if the clusters are close together, the posteriors will be very flat and EM can take longer than gradient descent methods to converge.

3.1 EM for MoG

As a concrete example, we derive the EM updates for fitting a mixture of Gaussians. Recall the MoG model

$$\mathbf{x}|z \sim \mathcal{N}(\boldsymbol{\mu}_z, \boldsymbol{\Sigma}_z), \quad p(z = k) =: \alpha_k$$

We define the parameter set $\boldsymbol{\theta}$ as the set of all $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \alpha_k$. Let $\mathbf{x}_1, ..., \mathbf{x}_N \in \mathbb{R}^d$ be our observed data. Define $q_{ki}^t := q^t(z_i = k|\mathbf{x}_i)$. The EM updates, derived below, are

E step:

$$q^{t+1}(z_i = k|\mathbf{x}_i) = \frac{\alpha_k^t p(\mathbf{x}_i|z_i = k; \boldsymbol{\theta}^t)}{\sum_{j=1}^K \alpha_j^t p(\mathbf{x}_i|z_i = j; \boldsymbol{\theta}^t)}$$

M step:

$$\boldsymbol{\mu}_k^{t+1} = \frac{\sum_{i=1}^N q_{ki}^{t+1} \mathbf{x}_i}{\sum_{i=1}^N q_{ki}^{t+1}}$$

$$\boldsymbol{\Sigma}_k^{t+1} = \frac{\sum_{i=1}^N q_{ki}^{t+1}(\mathbf{x}_i - \boldsymbol{\mu}_k^{t+1})(\mathbf{x}_i - \boldsymbol{\mu}_k^{t+1})^T}{\sum_{i=1}^N q_{ki}^{t+1}}$$

$$\alpha_k^{t+1} = \frac{1}{N}\sum_{i=1}^N q_{ki}^{t+1}$$

In the E step, we assign to each $\mathbf{x}_i$ a probability distribution over latents (that is, a soft assignment). This assignment is $p(\mathbf{x}_i|z_i = k; \boldsymbol{\theta}^t)$, the Gaussian likelihood of the data, but reweighted by the prior and normalized. In the M step, we are essentially computing the usual maximimum likelihood estimates of the parameters, but weighted by the posterior on z; indeed, if we set $q_{ki}^{t+1} = \frac{1}{N}$, then we are using the usual MLE. The update for $\boldsymbol{\mu}_k$ is entirely analogous to the update to the centroids for soft k-means (1). The main difference is that now we are also updating estimates of covariances and the prior and synthesizing this information in our posterior estimates in the E step, which will in turn influence the $\boldsymbol{\mu}_k$ assignments in the next E step.

We now derive these updates. Recall the log likelihood is

$$\ell(\boldsymbol{\theta}; \mathbf{x}) = \sum_{i=1}^N \log\left(\sum_{k=1}^K p(\mathbf{x}_i|z_i = k; \boldsymbol{\theta})p(z_i = k; \boldsymbol{\theta})\right)$$

For the E step, we set

$$\begin{aligned} q^{t+1}(z_i = k|\mathbf{x}_i) &= p(z_i = k|\mathbf{x}_i; \boldsymbol{\theta}^t) \\ &= \frac{p(z_i = k, \mathbf{x}_i; \boldsymbol{\theta}^t)}{p(\mathbf{x}_i; \boldsymbol{\theta}^t)} \\ &= \frac{p(\mathbf{x}_i|z_i = k; \boldsymbol{\theta}^t)p(z_i = k; \boldsymbol{\theta}^t)}{\sum_{j=1}^K p(\mathbf{x}_i, z_i = j; \boldsymbol{\theta}^t)} \\ &= \frac{p(\mathbf{x}_i|z_i = k; \boldsymbol{\theta}^t)p(z_i = k; \boldsymbol{\theta}^t)}{\sum_{j=1}^K p(\mathbf{x}_i|z_i = j; \boldsymbol{\theta}^t)p(z_i = j; \boldsymbol{\theta}^t)} \end{aligned}$$

For MoG, $p(\mathbf{x}|z = j; \boldsymbol{\theta}^t)$ is the pdf of $\mathcal{N}(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ evaluated at $\mathbf{x}$.

For the M step, we need to maximize the expected complete log-likelihood $\ell_q = \mathbb{E}_{q^{t+1}}(L_c(\mathbf{X}, \mathbf{z}; \boldsymbol{\theta}))$. The parameters to estimate are $\boldsymbol{\mu}_k$, $\boldsymbol{\Sigma}_k$, and α_k, the prior. The expected complete log likelihood is

$$\begin{aligned} \mathbb{E}_{q^{t+1}}(L_c(x, z; \boldsymbol{\theta})) &= \sum_{i=1}^N \sum_{k=1}^K q_{ki}^{t+1}\left[\log \alpha_k - \frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k) - \frac{1}{2}\log\Big((2\pi)^d|\boldsymbol{\Sigma}_k|\Big)\right] \\ &= \sum_{i=1}^N \sum_{k=1}^K q_{ki}^{t+1}\left[\log \alpha_k - \frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_k)^T\boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_i - \boldsymbol{\mu}_k) + \frac{1}{2}\log\Big((2\pi)^{-d}|\boldsymbol{\Sigma}_k^{-1}|\Big)\right] \end{aligned}$$

We can take gradients with respect to the parameters:

$$\frac{\partial \ell_q}{\partial \boldsymbol{\mu}_k} = \boldsymbol{\Sigma}_k^{-1}\sum_{i=1}^N q_{ki}^{t+1}(\mathbf{x}_i - \boldsymbol{\mu}_k)$$

$$\frac{\partial \ell_q}{\partial \boldsymbol{\Sigma}_k^{-1}} = \frac{1}{2}\sum_{i=1}^{N} q_{ki}^{t+1}[\boldsymbol{\Sigma}_k - (\mathbf{x}_i - \boldsymbol{\mu}_k^{t+1})(\mathbf{x}_i - \boldsymbol{\mu}_k^{t+1})^T]$$

Setting the gradients to zero and solving these equations gives us the updates for μ_k, Σ_k shown above. To obtain the update for α_k, we need to introduce the constraint $\sum_{k=1}^{K} \alpha_k = 1$ via Lagrange multipliers. We thus maximize the Lagrangian $\ell_q' = \mathbb{E}_{q^{t+1}}(L_c(x, z; \boldsymbol{\theta})) - \lambda(\sum_{k=1}^{K} \alpha_k - 1)$. Taking gradients, we get

$$\frac{\partial \ell_q'}{\partial \alpha_k} = -\lambda + \frac{1}{\alpha_k}\sum_{i=1}^{N} q_{ki}^{t+1} = 0$$

We can rearrange to write

$$\alpha_k \lambda = \sum_{i=1}^{N} q_{ki}^{t+1}$$

Summing over all k, we obtain

$$\begin{aligned}
\lambda \sum_{k=1}^{K} \alpha_k &= \sum_{k=1}^{K}\sum_{i=1}^{N} q_{ki}^{t+1} \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} q^{t+1}(z_i = k|\mathbf{x}_i) \\
&= \sum_{i=1}^{N} 1 = N
\end{aligned}$$

using the fact that q^{t+1} is a distribution over z_i. Since $\sum_{k=1}^{K} \alpha_k = 1$, the left hand side reduces to λ, so we conclude $\lambda = N$. Substituting into the original gradient, we have

$$\frac{\partial \ell_q'}{\partial \alpha_k} = -N + \frac{1}{\alpha_k}\sum_{i=1}^{N} q_{ki}^{t+1}$$

Setting this to zero gives us the desired updates for α_k.

3.2 Appendix: Jensen's Inequality Proof

Let f be convex, and X be a random variable. Construct the tangent line $L(X) = aX + b$ to f at $\mathbb{E}(X)$ for some a, b - this means $L(\mathbb{E}(X)) = f(\mathbb{E}(X))$. By convexity, $L(X) \leq f(X)$ for all X.[3] Then by monotonicity of expectation, we have

$$f(\mathbb{E}(X)) = L(\mathbb{E}(X)) = a\mathbb{E}(X) + b = \mathbb{E}(aX + b) = \mathbb{E}(L(X)) \leq \mathbb{E}(f(X))$$

[3] Actually, it takes some extra work to prove this intuitive fact. We will take it for granted here.

3.3 Appendix: Distance and Similarity

In our discussion of K-means, we restricted ourselves to using L^2 distance as a distance function. Formally, a distance function $d(x, y)$ is defined as a non-negative function that satisfies the following properties:

1. $d(x, y) = 0$ iff $x = y$
2. $d(x, y) = d(y, x)$ for all x, y
3. $d(x, z) \leq d(x, y) + d(y, z)$ for all x, y, z (triangle inequality)

A **dissimilarity** measure $d(x, y)$ is a function satisfying the above properties except possibly the triangle inequality. One possible **similarity** measure $s(x, y)$ can be defined as $-d(x, y)$. In clustering, we are free to choose our notion of similarity. Different algorithms may or may not work differently depending on which similarity measure we choose.

For example, it does not make sense to use the K-means algorithm if we care about L^1 distances instead of L^2. However, we can apply the same principles used to derive the K-means algorithm: if we replace the L^2 norm in the objective by L^1 and do a similar alternating minimization, then on the centroid assignment step, we will set each centroid to the median of the data instead of the mean. On the cluster assignment step, we will assign each point to the closest centroid in L^1 distance. This variation is called K-medians.

Deciding which similarity measure to use is a modeling choice that typically depends on the data and desired clustering properties. For example, K-medians may be of use if the data has outliers and we desire a more robust estimator of the clusters. In certain domains, such as computer vision, the L^p distances are not appropriate measures of dissimilarity, so other measures may be used.