Lecture 13 - Optimization of Gaussian Mixture Models with the EM Algorithm

The observed data likelihood for a Gaussian Mixture Model (GMM) is

$$\mathcal{L}^0 = \prod_{i=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(x_i|\mu_k, \Sigma_k)$$

What hidden variables can we add to simplify this problem?

- In this example, a hidden variable can be the label of the Gaussian from which x_i was drawn from.

$$z_i : \text{label of the Gaussian from which } x_i \text{ was drawn from}$$

If we know z_i, then for each data point we know which Gaussian it was drawn from along with its respective parameter μ_{z_i} and Σ_{z_i} and its respective weight π_{z_i}. So, each data point would had been drawn from $\pi_{z_i} N(x_i|\mu_{z_i}, \Sigma_{z_i})$.

Then, assuming we have $\{z_i\}_{i=1}^{N}$, we can write the complete data likelihood:

$$\mathcal{L}^c = \prod_{i=1}^{N} \pi_{z_i} \mathcal{N}(x_i|\mu_{z_i}, \Sigma_{z_i})$$

Now we can iterate between the **E-step** and **M-step** of the EM algorithm until we find convergence or we have reached a threshold for a number of iterations.

Optimization Function

We can now extent the optimization function:

$$\begin{aligned} Q(\Theta, \Theta^t) &= E[\ln(\mathcal{L}^c)|X, \Theta^t] \\ &= \sum_{\mathbf{z}} \ln(\mathcal{L}^c) P(z|X, \Theta^t) \\ &= \sum_{z_i=1}^{K} \ln(\mathcal{L}^c) P(\mathbf{z}_i|\mathbf{x}_i, \Theta^t) \end{aligned}$$

E-step (Expectation Step)

In order to complete the E-STEP, we need to know how to compute $P(\mathbf{z}_i|\mathbf{x}_i, \Theta^t)$.

- This is the posterior probability of the label z_i for data sample x_i.

- So we want to assign the label z_i to the data sample x_i for which the posterior probability is maximized (just like in Naive Bayes classification).

Recall from **Bayes' Rule**: for two non-empty events A and B, $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$. We can use this theorem to rewrite $P(\mathbf{z}_i|\mathbf{x}_i, \Theta^t)$:

$$\begin{aligned} P(\mathbf{z}_i|\mathbf{x}_i, \Theta^t) &= \frac{P(\mathbf{x}_i|\mathbf{z}_i, \Theta^t)P(\mathbf{z}_i|\Theta^t)}{P(\mathbf{x}_i|\Theta^t)} \\ &= \frac{P(\mathbf{x}_i|\mu_{z_i}^t, \Sigma_{z_i}^t)\pi_{z_i}^t}{\sum_{z_i=1}^K \pi_{z_i}^t P(\mathbf{x}_i|\mu_{z_i}^t, \Sigma_{z_i}^t)} \\ &= C_{ik} \end{aligned}$$

This is called the **memberships** or **responsabilities** matrix, which contains the label assignment for point x_i in each Gaussian component $k.

- In the E-STEP we estimate the membership matrix $C_{ik} = P(\mathbf{z}_i|\mathbf{x}_i, \Theta^t)$. This matrix is of size $N \times K$ that contains the likelihoods of each point belonging in each one of the Gaussians.
- A good check when implementing this matrix is to make sure that the sum of the rows are equal to 1!

This completes the Expectation step (E-step) in EM. Now, we derive the update equations for the parameters $\Theta = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^K$ in the Maximization step.

M-step (Maximization Step)

In the **M-step**, we are going to use (and hold constant) the membership matrix C_{ik} we learned from the E-step.

We will know estimate the new set of parameters $\Theta = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^K$ that maximize $Q(\Theta, \Theta^t)$, i.e.

$$\arg_{\Theta} \max Q(\Theta, \Theta^t)$$

Without loss of generality, let's assume that the covariance matrices are isotropic: $\Sigma_k = \sigma_k^2 \mathbf{I}$, then we can rewrite it as:

$$
\begin{aligned}
Q(\Theta, \Theta^t) &= \sum_{z_i=1}^{K} \ln(\mathcal{L}^c) P(\mathbf{z}_i|\mathbf{x}_i, \Theta^t) \\
&= \sum_{z_i=1}^{K} \ln\left(\prod_{i=1}^{N} \pi_{z_i} \mathcal{N}(x_i|\mu_{z_i}, \Sigma_{z_i})\right) P(\mathbf{z}_i|\mathbf{x}_i, \Theta^t) \\
&= \sum_{k=1}^{K} \ln\left(\prod_{i=1}^{N} \pi_{k} \mathcal{N}(x_i|\mu_{k}, \Sigma_{k})\right) P(\mathbf{z}_i = k|\mathbf{x}_i, \Theta^t) \\
&= \sum_{k=1}^{K} \sum_{i=1}^{N} \left(\ln(\pi_k) + \ln\big(\mathcal{N}(x_i|\mu_k, \Sigma_k)\big)\right) C_{ik} \\
&= \sum_{k=1}^{K} \sum_{i=1}^{N} \left(\ln(\pi_k) - \frac{d}{2}\ln(2\pi) - \frac{d}{2}\ln(\sigma_k^2) - \frac{1}{2\sigma_k^2}\|\mathbf{x}_i - \mu_k\|_2^2\right) C_{ik}
\end{aligned}
$$

So, now we want to solve:

$$
\begin{aligned}
\frac{\partial Q(\Theta, \Theta^t)}{\partial \mu_k} &= 0 \\
\frac{\partial Q(\Theta, \Theta^t)}{\partial \sigma_k^2} &= 0 \\
\frac{\partial Q(\Theta, \Theta^t)}{\partial \pi_k} &= 0
\end{aligned}
$$

Solving for μ_k

$$
\begin{aligned}
0 &= \frac{\partial Q(\Theta, \Theta^t)}{\partial \mu_k} \\
0 &= \sum_{i=1}^{N} \left(\frac{1}{\sigma_k^2}(\mathbf{x}_i - \mu_k)\right) C_{ik} \\
0 &= \sum_{i=1}^{N} (\mathbf{x}_i - \mu_k) C_{ik} \\
0 &= \sum_{i=1}^{N} \mathbf{x}_i C_{ik} - \sum_{i=1}^{N} \mu_k C_{ik} \\
\sum_{i=1}^{N} \mathbf{x}_i C_{ik} &= \mu_k \sum_{i=1}^{N} C_{ik} \\
\mu_k &= \frac{\sum_{i=1}^{N} \mathbf{x}_i C_{ik}}{\sum_{i=1}^{N} C_{ik}}
\end{aligned}
$$

- We can interpret this as a **weighted mean**, where the weights represent the likelihood of each point belonging to cluster k.

Solving for σ_k^2

$$
\begin{aligned}
0 &= \frac{\partial Q(\Theta, \Theta^t)}{\partial \sigma_k^2} \\
0 &= \sum_{i=1}^{N} \left(-\frac{d}{2\sigma_k^2} + \frac{2}{(2\sigma_k^2)^2} \|x_i - \mu_k\|_2^2 \right) C_{ik} \\
\frac{d}{2\sigma_k^2} \sum_{i=1}^{N} C_{ik} &= \frac{2}{(2\sigma_k^2)^2} \sum_{i=1}^{N} C_{ik} \|x_i - \mu_k\|_2^2 \\
d\sigma_k^2 \sum_{i=1}^{N} C_{ik} &= \sum_{i=1}^{N} C_{ik} \|x_i - \mu_k\|_2^2 \\
\sigma_k^2 &= \frac{\sum_{i=1}^{N} C_{ik} \|x_i - \mu_k\|_2^2}{d \sum_{i=1}^{N} C_{ik}}
\end{aligned}
$$

- Again, assuming that the covariance of each cluster is isotropic, that is, $\Sigma_k = \sigma_k^2 \mathbf{I}$ (where Σ_k in a $d \times d$ matrix.)
- For the variance, this can be interpreted as how much each data point contributes to the k-th cluster.

Solving for π_k

For π_k we have the contraint that $\sum_{k=1}^{K} \pi_k = 1$. Then we need to incorporate it into $Q(\Theta, \Theta^t)$ using **Lagrange Multipliers**.

$$
Q_\pi(\Theta, \Theta^t) = Q(\Theta, \Theta^t) + \lambda \left(1 - \sum_{k=1}^{K} \pi_k \right)
$$

where λ is called the Lagrange multiplier.

$$
\begin{aligned}
0 &= \frac{\partial Q_\pi(\Theta, \Theta^t)}{\partial \pi_k} \\
0 &= \sum_{i=1}^{N} C_{ik} \frac{1}{\pi_k} - \lambda \\
\pi_k &= \frac{\sum_{i=1}^{N} C_{ik}}{\lambda}
\end{aligned}
$$

Since $\sum_k \pi_k = 1$, we have that:

$$
\begin{aligned}
\sum_{k=1}^{K} \pi_k &= 1 \\
\sum_{k=1}^{K} \frac{\sum_{i=1}^{N} C_{ik}}{\lambda} &= 1 \\
\lambda &= \sum_{i=1}^{N} \sum_{k=1}^{K} C_{ik}
\end{aligned}
$$

Pluggin it back, we find:

$$\pi_k = \frac{\sum_{i=1}^{N} C_{ik}}{\lambda}$$

$$\pi_k = \frac{\sum_{i=1}^{N} C_{ik}}{\sum_{i=1}^{N}\sum_{k=1}^{K} C_{ik}}$$

$$= \frac{\sum_{i=1}^{N} C_{ik}}{\sum_{i=1}^{N} 1}$$

$$= \frac{\sum_{i=1}^{N} C_{ik}}{N}$$

- The π_k's are summing up all its responsabilites over all data points.

Pseudo-Code

We now have everything we need to implement the EM algorithm for Gaussian Mixtures.

- The pseudo-code for the algorithm is:

In [1]:
```
from IPython.display import Image
Image('figures/PseudoCode_EMforGMM.png',width=700)
```

Out[1]:

Algorithm 1 EM for Gaussian Mixture Model

INPUT: Training data $\mathbf{X}$, number of Gaussian terms K
Initialize all parameters (μ_k, Σ_k and π_k)
t=1
while convergence not yet reached OR maximum number of iterations reached **do**
  E-STEP:
  Compute $C_{ik} = \frac{\pi_{z_i}^t P(\mathbf{x}_i|\mu_{z_i}^t, \Sigma_{z_i}^t)}{\sum_{z_i=1}^K \pi_{z_i}^t P(\mathbf{x}_i|\mu_{z_i}^t, \Sigma_{z_i}^t)}$ for every x_i and k.
  C is a $N \times k$ matrix, where each row sums to 1
  M-STEP:
  (1) Update μ_k for all k. $\mu_k^{t+1} = \frac{\sum_{i=1}^N C_{ik} x_i}{\sum_{i=1}^N C_{ik}}$, where μ_k is a $d \times 1$, and U is a $d \times k$ matrix.
  (2) Update σ_k^2 for all k. $\sigma_k^{2^{t+1}} = \frac{\sum_{i=1}^N C_{ik} \|x_i - \mu_k^t\|_2^2}{d \sum_{i=1}^N C_{ik}}$, where σ_k^2 is a $d \times d$, and Σ is a $d \times d \times k$ tuple.
  (3) Update π_k for all k. $\pi_k^{t+1} = \frac{\sum_{i=1}^N C_{ik}}{N}$, where π_k is a scalar, and Π is a $d \times 1$ vector.
  t = t + 1
  Check convergence criteria
end while
OUTPUT: C_{ik}, μ_k, Σ_k and π_k

Discussions

- Does the EM algorithm find the **global minima**?
- Given a data set with an unknown number of groups/clusters, can you come up with a strategy for determining the "right" number of groups?

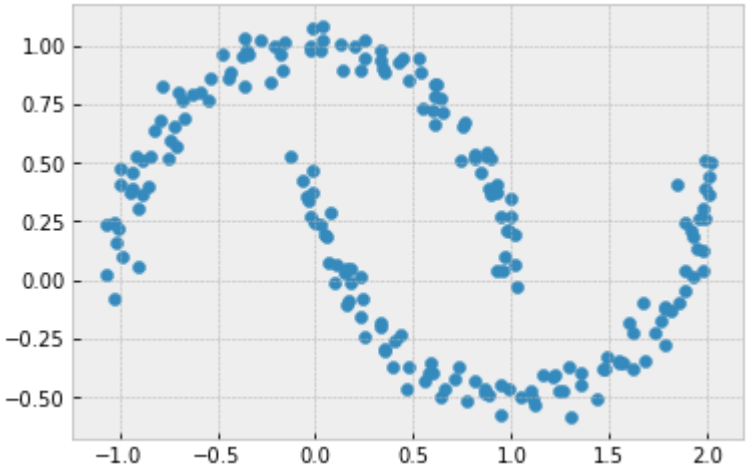
Example: GMM as Density Estimation

Consider some data generated from Scikit-Learn's `make_moons` function:

In [4]:

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('bmh')
from scipy import stats

from sklearn.datasets import make_moons
from matplotlib.patches import Ellipse
```

```
Xmoon, ymoon = make_moons(200, noise=.05, random_state=0)
plt.scatter(Xmoon[:, 0], Xmoon[:, 1]);
```



Let's create a helper function that will help us visualize the locations and shapes of the GMM clusters by drawing ellipses based on the GMM output.

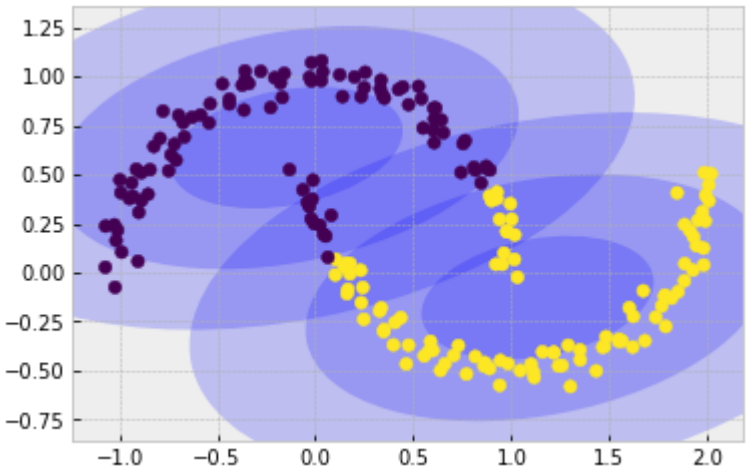
- Let's not worry about the implementation of this helper function.

In [5]:
```
# Code from "Python Data Science Handbook"
# https://jakevdp.github.io/PythonDataScienceHandbook/05.12-gaussian-mixtures.html

def draw_ellipse(position, covariance, ax=None, **kwargs):
    """Draw an ellipse with a given position and covariance"""
    ax = ax or plt.gca()

    # Convert covariance to principal axes
    if covariance.shape == (2, 2):
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 2 * np.sqrt(covariance)

    # Draw the Ellipse
    for nsig in range(1, 4):
        ax.add_patch(Ellipse(position, nsig * width, nsig * height,
                             angle, **kwargs))

def plot_gmm(gmm, X, label=True, ax=None):
    ax = ax or plt.gca()
    labels = gmm.fit(X).predict(X)
    if label:
        ax.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis', zorder=2)
    else:
        ax.scatter(X[:, 0], X[:, 1], s=40, zorder=2)
    ax.axis('equal')

    w_factor = 0.2 / gmm.weights_.max()
    for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):
        draw_ellipse(pos, covar, alpha=w * w_factor)
```

If we try to fit this with a two-component GMM viewed as a clustering model, the results are not particularly useful:

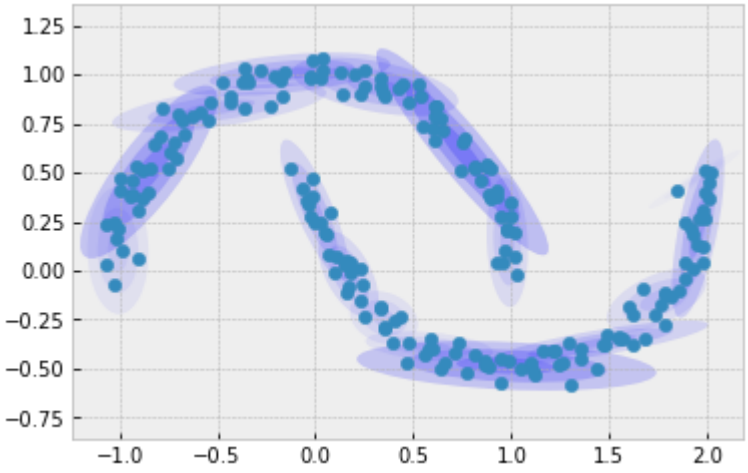
In [6]:
```
from sklearn.mixture import GaussianMixture

GaussianMixture?
```

In [7]:
```
GMM = GaussianMixture(n_components=2, covariance_type='full', random_state=0).fit(Xmoo
plot_gmm(GMM, Xmoon)
```



But if we instead use many more components and ignore the cluster labels, we find a fit that is much closer to the input data:

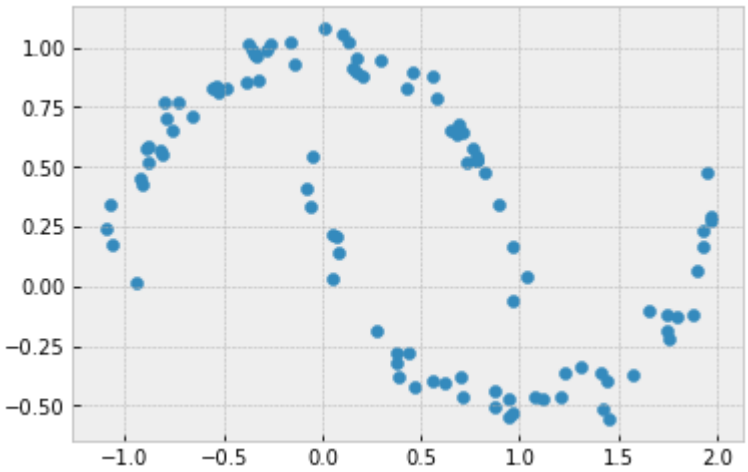
In [8]:
```
GMM16 = GaussianMixture(n_components=16, random_state=0).fit(Xmoon)
plot_gmm(GMM16, Xmoon, label=False)
```



Here the mixture of 16 Gaussians serves not to find separated clusters of data, but rather to model the overall distribution of the input data. This is a generative model of the distribution, meaning that the GMM gives us the recipe to generate new random data distributed similarly to our input.

For example, here are 400 new points drawn from this 16-component GMM fit to our original data:

In [9]:
```
Xnew = GMM16.sample(100)
plt.scatter(Xnew[0][:, 0], Xnew[0][:, 1]);
```



GMM is convenient as a flexible means of modeling an arbitrary multi-dimensional distribution of data.
