Lecture 14 - GMMs continued

In [3]:
```
from scipy.stats import multivariate_normal
import numpy as np
import numpy.random as npr

import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('seaborn-colorblind')

from IPython.display import Image
```

Gaussian Mixture Models

A **Gaussian Mixture Model** or **GMM** is a probabilistic model that assumes a data likelihood to be a weighted sum of Gaussian distributions with unknown parameters.

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k N(\mathbf{x}|\mu_k, \Sigma_k)$$

where $0 \leq \pi_k \leq 1$ and $\sum_{k=1}^{K} \pi_k = 1$.

Optimizing GMM 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.

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

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

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

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

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

M-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.

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

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

Solving for μ_k

From last class:

$$\mu_k = \frac{\sum_{i=1}^{N} \mathbf{x}_i C_{ik}}{\sum_{i=1}^{N} C_{ik}}$$

- 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

From last class:

$$\sigma_k = \frac{\sum_{i=1}^{N} C_{ik} \|x_i - \mu_k\|_2^2}{\sum_{i=1}^{N} C_{ik}}$$

- Again, assuming that the covariance of each cluster is isotropic, that is, $\Sigma_k = \sigma_k^2 \mathbf{I}$.
- 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 $\sum_k \pi_k = 1$, then:

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

So:

$$\begin{aligned} \pi_k &= \frac{\sum_{i=1}^{N} C_{ik}}{\lambda} \\ \pi_k &= \frac{\sum_{i=1}^{N} C_{ik}}{\sum_{i=1}^{N} \sum_{k=1}^{K} C_{ik}} \\ &= \frac{\sum_{i=1}^{N} C_{ik}}{\sum_{i=1}^{N} 1}, \text{ because } C_{ik} = \sum_{k=1}^{K} P(z_i = k | x_i, \Theta^t) \text{ is a (discrete) distribution} \\ &= \frac{\sum_{i=1}^{N} C_{ik}}{N} \end{aligned}$$

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

Implementation - Pseudo-Code

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

- The pseudo-code for the algorithm is:

In [3]:
```
Image('figures/PseudoCode_EMforGMM.png',width=700)
```

Out[3]:

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. $\sigma_k^{2^{t+1}} = \frac{\sum_{i=1}^N C_{ik} \|x_i - \mu_k^t\|_2^2}{\sum_{i=1}^N C_{ik}}$, 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

Alternating Optimization

- 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 Data Likelihood Estimation or Clustering

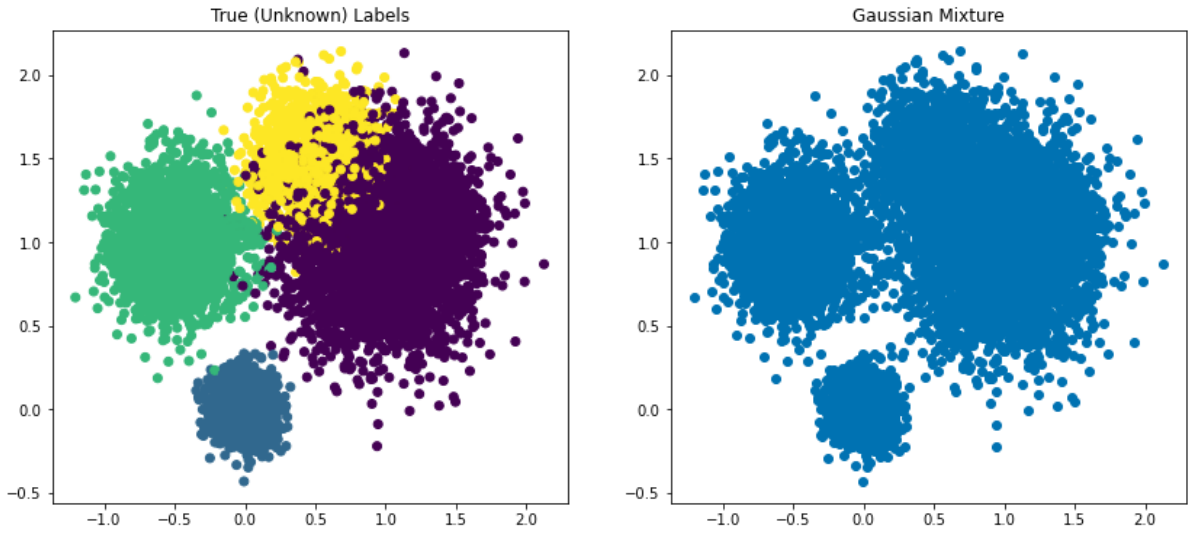
GMM is commonly used as an algorithm for density estimation. That is to say, the result of a GMM fit to some data is technically not a clustering model, but a generative probabilistic model describing the distribution of the data.

However, a common practical use for the GMM is as a clustering algorithm, where we are interested in finding groups in the data.

Let's generate some data:

In [1]:
```
def make_GaussianMixture(N, Means, Sigs, Pis):
    X = np.empty((0,Means.shape[1]))
    L = np.empty(0)
    for i in range(N):
        rv = npr.uniform()   # sample uniform RV
        GaussianChosen = np.where(rv < np.cumsum(Pis))[0][0]
        L = np.append(L, GaussianChosen)
        X = np.append(X, np.array([npr.multivariate_normal(Means[GaussianChose
n],
                                                           np.eye(Means.shape[
1])*Sigs[GaussianChosen])]), axis=0)
    return X, L
```

In [4]:
```
N = 10000
Means = np.array([[1,1],[0,0],[-.5, 1],[.5, 1.5]])
Sigs = [.1, .01, .05, .05]
Pis = [.4, .25, .25, .1]

X,L = make_GaussianMixture(N, Means, Sigs, Pis)

fig = plt.figure(figsize=(14,6))
fig.add_subplot(1,2,1)
plt.scatter(X[:,0],X[:,1], c=L)
plt.title('True (Unknown) Labels');
fig.add_subplot(1,2,2)
plt.scatter(X[:,0],X[:,1])
plt.title('Gaussian Mixture');
```



The picture on the left, represents the entire data color-coded according to a class they belong to.

In practice we are **not** given labels, and so we work directly with the unlabeled data set on the right.

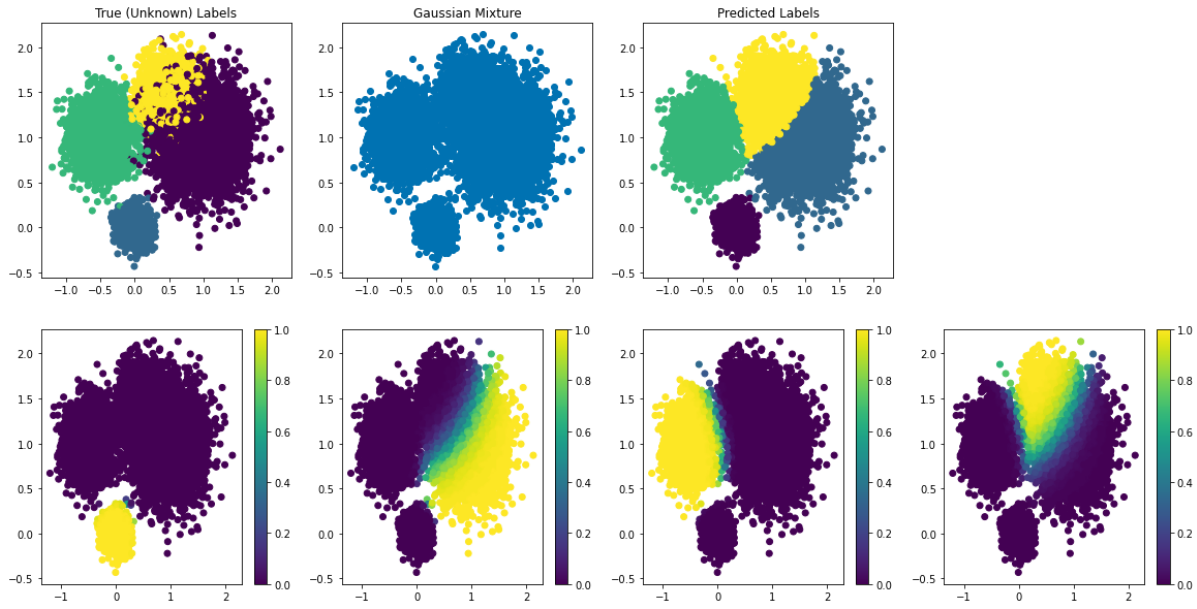
- We can use GMM to uncover groups in the data.

To illustrate this, I will use the `scikit-learn` algorithm implementation of the GMM model:

In [5]:
```
from sklearn.mixture import GaussianMixture
```

In [6]:
```
?GaussianMixture
```

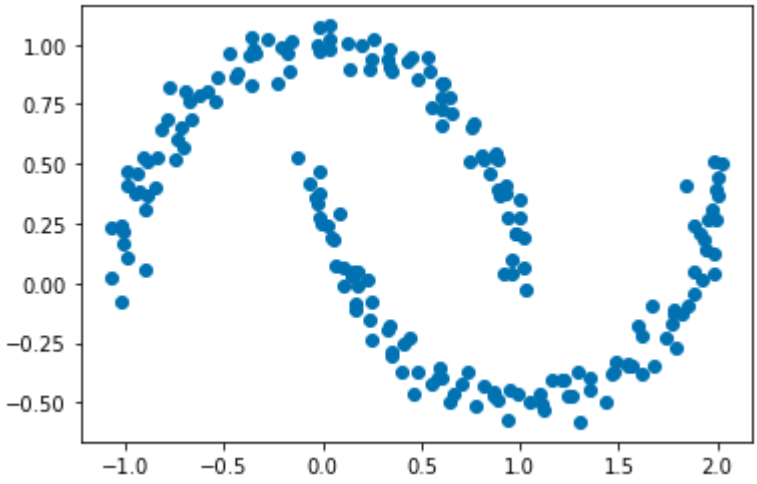
In [9]:
```
Nclusters = 4
GMM = GaussianMixture(n_components=Nclusters).fit(X)
labels = GMM.predict(X)
prob = GMM.predict_proba(X).round(2)

fig = plt.figure(figsize=(20,10))
fig.add_subplot(2,Nclusters,1)
plt.scatter(X[:,0],X[:,1], c=L)
plt.title('True (Unknown) Labels');
fig.add_subplot(2,Nclusters,2)
plt.scatter(X[:,0],X[:,1])
plt.title('Gaussian Mixture');
fig.add_subplot(2,Nclusters,3)
plt.scatter(X[:,0],X[:,1], c=labels)
plt.title('Predicted Labels');
for i in range(Nclusters):
    ax = fig.add_subplot(2,Nclusters,Nclusters+i+1)
    p1 = ax.scatter(X[:,0], X[:,1], c=prob[:,i])
    fig.colorbar(p1, ax=ax)
```



Example: GMM as Density Estimation

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

In [20]:
```
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]);
```



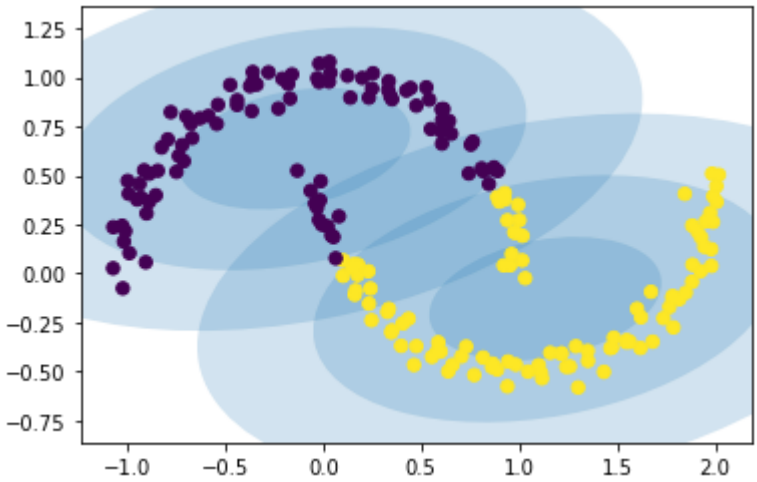
Let's create a helper function that will help us visualize the locations and shapes of the GMM clusters by drawing ellipses based on the GMM output.

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

In [21]:
```
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)
```

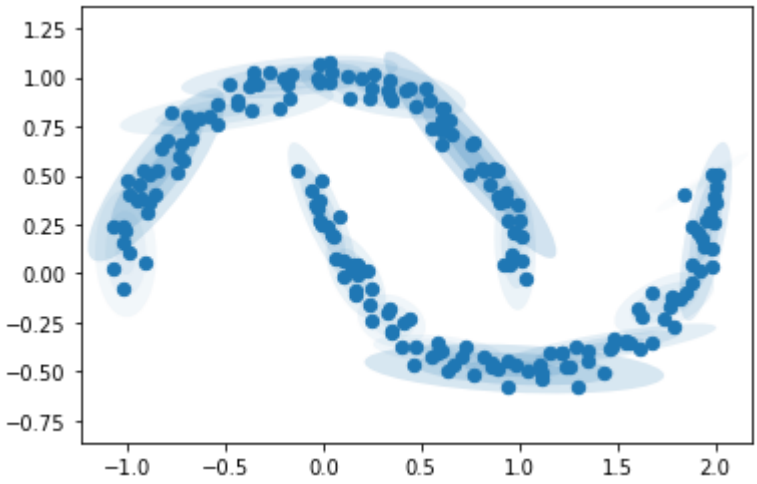
If we try to fit this with a two-component GMM viewed as a clustering model, the results are not particularly useful:

In [8]:
```
GMM = GaussianMixture(n_components=2, covariance_type='full', random_state=0).
fit(Xmoon)
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:

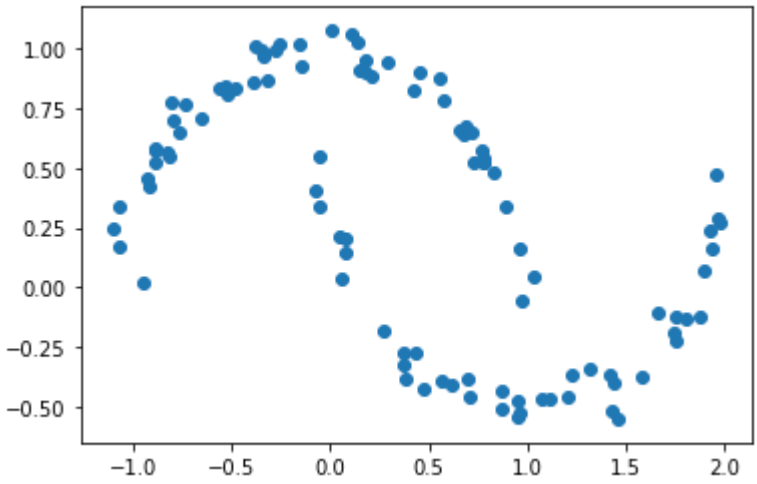
In [9]:
```
GMM16 = GaussianMixture(n_components=16, covariance_type='full', random_state=
0).fit(Xmoon)
plot_gmm(GMM16, Xmoon, label=False)
```



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 [11]:
```
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.

In [ ]: