Lab 1

Gaussian Mixture Models

Lab Objective: *Understand the formulation of Gaussian Mixture Models (GMMs) and how to estimate GMM parameters.*

You've already seen GMMs as the observation distribution in certain continuous density HMMs. Here, we will discuss them further and learn how to estimate their parameters, given data.

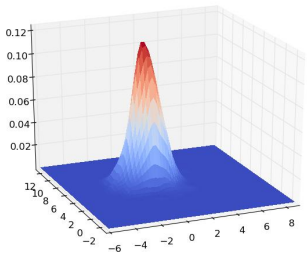
The main idea behind a mixture model is contained in the name, i.e. it is a *mixture* of different models. What do we mean by a mixture? A mixture model is composed of K *components*, each component being responsible for a portion of the data. The responsibilities of these components are represented by mixture *weights* w_i, for $i = 1, \cdots, k$. As you may have guessed, these weights are nonnegative and sum to 1. Thus component j is responsible for $100 \cdot w_j$ percent of the data generated by the model.

Each component is itself a probability distribution. In a GMM, each component is specifically a Gaussian (multivariate normal) distribution. Thus we additionally have parameters μ_i, Σ_i for $i = 1, \cdots, K$, i.e. a mean and covariance for each component in the GMM. It is important here to keep in mind that a GMM does not arise from adding weighted multivariate normal random variables, but rather from weighting the responsibility of each multivariate normal random variable. In the first case, we would simply have a different multivariate normal distribution, whereas in the second case we have a mixture. Refer to Figure **??** for a visualization of this.

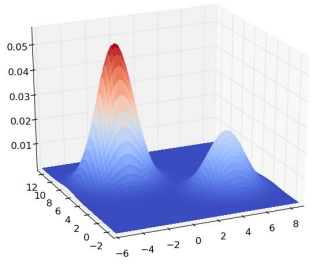
Thus, a fully defined GMM has parameters $\lambda = (w, \mu, \mathbf{\Sigma})$. The density of a GMM is given by $\mathbb{P}(x|\lambda) = \sum_{i=1}^{K} w_i \mathcal{N}(x; \mu_i, \Sigma_i)$ where

$$\mathcal{N}(x; \mu_i, \Sigma_i) = \frac{1}{(2\pi)^{\frac{K}{2}} |\Sigma_i|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1} (x-\mu_i)}$$

Problem 1. Write a function to evaluate the density of a normal distribution at a point x, given parameters μ and Σ. Include the option to return the log of this probability, but be sure to do it intelligently! Also write a function



(a) Sum of weighted multivariate normal random variables.



(b) Weighted mixture of multivariate normal random variables.

that computes the density of a GMM at a point x, given the parameters λ, along with the log option.

Throughout this lab, we will build a GMM class with various methods. We will outline this now.

Problem 2. Write the skeleton of a GMM class. In the `__init__` method, it should accept the non-null parameter *n_components*, as well as parameters for the weights, means, and covariance matrices which define the GMM. Include a function to generate data from a fully defined GMM (you may use your code from the CDHMM lab for this), as well as the density function you recently defined.

The main focus of this lab will be to estimate the parameters of a GMM, given observed multivariate data $Y = y_1, y_2, \cdots, y_T$. This can be done via Gibbs sampling, as well as with EM (Expectation Maximization). We choose the latter approach for this lab. To do this, we must compute the probability of an observation being from each component of a GMM with parameters $\lambda^{(n)} = \left(w^{(n)}, \mu^{(n)}, \Sigma^{(n)}\right)$. This is simply

$$\mathbb{P}(x_t = i | y_t, \lambda) \propto w_i^{(n)} \mathcal{N}(y_t; \mu_i^{(n)}, \Sigma_i^{(n)})$$

Just as with HMMs, we refer to these probabilities as $\gamma_t(i)$, and this is the E-step in the algorithm. This might seem straightforward, except this direct computation will likely lead to numerical issues. Instead, we work in the log space, which means we have to be a bit more careful.

It is feasible (and occurs quite often) that each term $w_i^{(n)} \mathcal{N}(y_t; \mu_i^{(n)}, \Sigma_i^{(n)})$ is 0, because of underflow in the computation of the multivariate normal density. Letting $l_i^{(n)} = \ln w_i^{(n)} + \ln \mathcal{N}(y_t; \mu_i^{(n)}, \Sigma_i^{(n)})$, we can compute these probabilities

more carefully, as follows:

$$\begin{aligned}\mathbb{P}(x_t = i|y_t, \lambda) &= \frac{e^{l_i}}{\sum_{j=1}^{K} e^{l_j}} \\ &= \frac{e^{l_i} e^{-\max_k l_k}}{\sum_{j=1}^{K} e^{l_j} e^{-\max_k l_k}} \\ &= \frac{e^{l_i - \max_k l_k}}{\sum_{j=1}^{K} e^{l_j - \max_k l_k}}\end{aligned}$$

which will effectively avoid underflow problems.

Problem 3. Add a method to your class to compute $\gamma_t(i)$ for $t = 1, \cdots, T$ and $i = 1, \cdots, K$. Don't forget to do this intelligently to avoid underflow!

Given our matrix γ, we can reestimate our weights, means, and covariance matrices as follows:

$$\begin{aligned}w_i^{(n+1)} &= \sum_{t=1}^{T} \gamma_t(i) \\ \mu_i^{(n+1)} &= \frac{\sum_{t=1}^{T} \gamma_t(i) y_t}{\sum_{t=1}^{T} \gamma_t(i)} \\ \Sigma_i^{(n+1)} &= \frac{\sum_{t=1}^{T} \gamma_t(i)(y_t - \mu_i^{(n+1)})(y_t - \mu_i^{(n+1)})^T}{\sum_{t=1}^{T} \gamma_t(i)}\end{aligned}$$

for $i = 1, \cdots, K$. These updates are the M-step in the algorithm.

Problem 4. Add methods to your class to update w, μ and Σ as described above.

With the above work, we are almost ready to complete our class. To train, we will randomly initialize our parameters λ, and then iteratively update them as above.

Problem 5. Add a method to initialize λ. Do this intelligently, i.e. your means should not be far from your actual data used for training, and your covariances should neither be too big nor too small. Your weights should roughly be equal, and still sum to 1. Also add a method to train your model, as described previously, iterating until convergence within some tolerance.

We will use our work to train the "Mickey Mouse" GMM, which has parameters

$$
\begin{aligned}
w &= \begin{bmatrix} 0.7 & 0.15 & 0.15 \end{bmatrix} \\
\mu_1 &= \begin{bmatrix} 0.0 & 0.0 \end{bmatrix} \\
\mu_2 &= \begin{bmatrix} -1.5 & 2.0 \end{bmatrix} \\
\mu_3 &= \begin{bmatrix} 1.5 & 2.0 \end{bmatrix} \\
\Sigma_1 &= I_3 \\
\Sigma_2 &= 0.25 \cdot I_3 \\
\Sigma_3 &= 0.25 \cdot I_3
\end{aligned}
$$

To look at this GMM, we will evaluate the density at each point on a grid, as follows:

```
>>> import matplotlib.pyplot as plt
>>> x = np.arange(-3, 3, 0.1)
>>> y = np.arange(-2, 3, 0.1)
>>> X, Y = np.meshgrid(x, y)
>>> N, M = X.shape
>>> immat = np.array([[model.dgmm(np.array([X[i,j],Y[i,j]])) for j in xrange(M)] ↩
    for i in xrange(N)])
>>> plt.imshow(immat, origin='lower')
>>> plt.show()
```

See Figure 1.2 for this plot.

Problem 6. Generate 750 samples from the above mixture model. Using just the drawn samples, retrain your model. Evaluate and plot your density on the grid used above. How similar is your density to the original?

How close is our trained model to the original one? We can use the symmetric Kullback-Liebler divergence to measure the distance between two probability distributions with densities $p(x)$ and $p'(x)$:

$$
SKL(p, p') = \left| \frac{1}{2} \int p(x) \ln \frac{p(x)}{p'(x)} dx + \frac{1}{2} \int p'(x) \ln \frac{p'(x)}{p(x)} dx \right|
$$

We cannot analytically compute this, so we use a Monte Carlo approximation, which uses the fact that

$$
\frac{1}{N} \sum_{i=1}^{N} f(x_i) \to \int f(x) p(x) dx
$$

as $N \to \infty$, assuming that each $x_i \sim p$. Then we have the following approximation of the symmetric KL divergence:

$$
SKL(p, p') \approx \frac{1}{2N} \left| \sum_{i=1}^{N} \ln \frac{p(x_i)}{p'(x_i)} + \sum_{i=1}^{N} \ln \frac{p'(x'_i)}{p(x'_i)} \right|
$$

where $x_i \sim p$ and $x'_i \sim p'$, for large N.

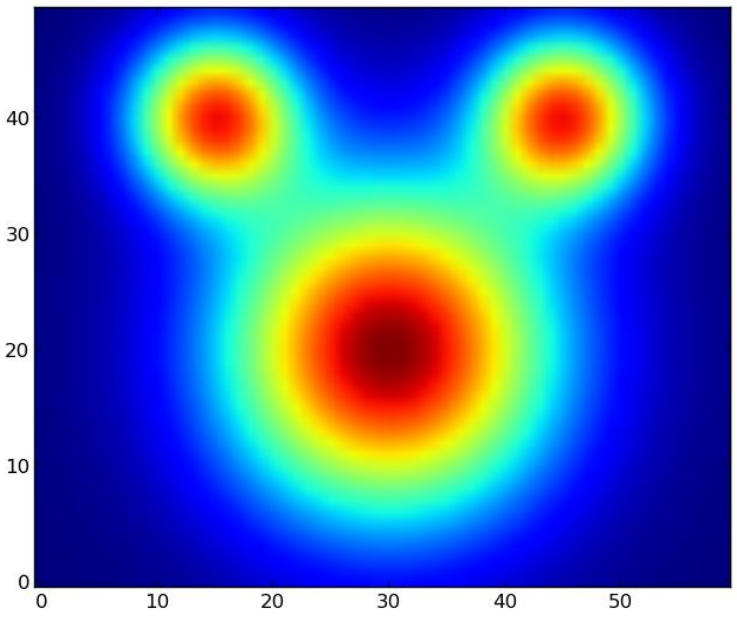


Figure 1.2: Density of true "Mickey Mouse" GMM.

Problem 7. Write a function to compute the approximate the SKL of two GMMs. Compute the SKL between a randomly initialized GMM and the known GMM. Compute the SKL between the trained GMM and the known GMM. Is our trained model a good fit?