# exercise_07

February 3, 2021

## CS-E4820 Machine Learning: Advanced Probabilistic Methods (spring 2021)

Pekka Marttinen, Santosh Hiremath, Tianyu Cui, Yogesh Kumar, Zheyang Shen, Alexander Aushev, Khaoula El Mekkaoui, Shaoxiong Ji, Alexander Nikitin, Sebastiaan De Peuter, Joakim Järvinen.

### Exercise 7, due on Tuesday March 23 at 23:00.

### Problem 1: ELBO for the simple model (1/2)

This problem and the next deal with deriving the ELBO for the 'simple model', described in the PDF document 'simple_elbo.pdf'. Before doing these exercises, familiarize yourself with the contents of the document.

**(a)** Show that the general formula for ELBO, shown in Equation (8), can be written as the sum shown in Equation (9) for the simple model.

**(b)** Derive the 2nd term $E_{q(\theta)}[logp(\theta)]$ of the ELBO. (**Hint:** recall that $Var(X) = E(X^2) - E(X)^2$).

**(c)** Find out the formula for the 7th term $E_{q(\theta)[logq(\theta)]}$ of the ELBO. (**Hint:** see the 6th term)

#### Solution

#### ELBO for simple model

Derivation of the ELBO for the simple model.(P. Marttinen)

Recall that variational inference is based on the decomposition

$$\log p(x) = \mathcal{L}(q) + KL(q|p)$$

where $q(\mathbf{Z})$ is any approximation to the posterior distribution $p(\mathbf{Z}|\mathbf{X})$ of unobserved variables $\mathbf{Z}$ in the model, given observed variables $\mathbf{X}$. The goal of the variational inference algorithm is to maximize the evidence lower bound (ELBO) $\mathcal{L}(q)$, or equivalently minimize the KL-divergence $KL(q|p)$ between the approximation and the true posterior. Here we show how to compute the ELBO for the 'simple model' derived earlier[1]. Briefly, the model is

$$p(x_n|\theta,\tau) = (1-\tau)N(x_n|0,1) + \tau N(x_n|\theta,1), \quad n = 1,\ldots,N.$$

The latent variable representation is given by

[1]The derivation of the ELBO for the general GMM case can be found in Bishop's book, Section 10.2.2.

$$p(\mathbf{x}|\mathbf{z},\theta) = \prod_{n=1}^{N} N(x_n|0,1)^{z_{n1}} N(x_n|\theta,1)^{z_{n2}}, \tag{1}$$

and

$$p(\mathbf{z}|\tau) = \prod_{n=1}^{N} \tau^{z_{n2}}(1-\tau)^{z_{n1}}. \tag{2}$$

Priors are specified as follows:

$$\begin{aligned} p(\tau) &= Beta(\tau \mid \alpha_0, \alpha_0) \propto \tau^{\alpha_0 - 1}(1-\tau)^{\alpha_0-1} \\ p(\theta) &= N(\theta \mid 0, \beta_0^{-1}) \propto \exp\left(-\frac{\beta_0}{2}\theta^2\right). \end{aligned}$$

The logarithm of the joint distribution can be written as:

$$\log p(\mathbf{x},\mathbf{z},\tau,\theta) = \log p(\tau) + \log p(\theta) + \log p(\mathbf{z}|\tau) + \log p(\mathbf{x} \mid \mathbf{z},\theta). \tag{3}$$

We assume the mean-field approximation

$$p(\mathbf{z},\tau,\theta \mid \mathbf{x}) \approx q(\tau)q(\theta)\textstyle\prod_n q(z_n). \tag{4}$$

Assume that currently we have factors

$$\begin{aligned} q(z_n \mid r_{n1}, r_{n2}) &= Categorical(z_n \mid r_{n1}, r_{n2}) = r_{n1}^{z_{n1}} r_{n2}^{z_{n2}}, & (5)\\ q(\tau) &= Beta(\tau \mid \alpha_\tau, \beta_\tau), & (6)\\ q(\theta) &= N(\theta \mid m_2, \beta_2^{-1}), & (7) \end{aligned}$$

where $r_{n1}, r_{n2}, n = 1, \ldots, N$, and so-called *variational parameters* are $\alpha_\tau, \beta_\tau, m_2$, and $\beta_2$, i.e. parameters that specify the exact distribution of the factor. Previously, in document 'simple_vb_example.pdf', we derived formulas for updating the the variational parameters of any factor conditionally on the other factors.

The general formula for the ELBO is given by

$$\begin{aligned} \mathcal{L}(q) &= \int q(\mathbf{Z}) \log \frac{p(\mathbf{X},\mathbf{Z})}{q(\mathbf{Z})} d\mathbf{Z} & (1)\\ &= E_q\left[\log p(\mathbf{X},\mathbf{Z})\right] - E_q\left[\log q(\mathbf{Z})\right], & (8) \end{aligned}$$

where $\mathbf{Z}$ is a generic notation that includes all unobservables. In the case of the simple model, defined by equations (1) and (2), $\mathbf{Z} = (\mathbf{z}, \tau, \theta)$, and the ELBO can be written as

$$\mathcal{L}(q) = E_q\left[\log p(\mathbf{x}, \mathbf{z}, \tau, \theta)\right] - E_q\left[\log q(\mathbf{z}, \tau, \theta)\right].$$

Using the factorization (3) and the mean-field approximation (4), the above equation can be written as:

$$\begin{aligned}\mathcal{L}(q) = {} & E_{q(\tau)}\left[\log p(\tau)\right] + E_{q(\theta)}\left[\log p(\theta)\right] + E_{q(\mathbf{z})q(\tau)}\left[\log p(\mathbf{z}|\tau)\right] + E_{q(\mathbf{z})q(\theta)}\left[\log p(\mathbf{x}|\mathbf{z}, \theta)\right] && (2)\\ & - E_{q(\mathbf{z})}\left[\log q(\mathbf{z})\right] - E_{q(\tau)}\left[\log q(\tau)\right] - E_{q(\theta)}\left[\log q(\theta)\right]. && (9)\end{aligned}$$

When conjugate priors are used, as is the case with the simple model, all seven terms in formula (9) can be computed analytically. Below we consider each of these terms in turn. \textbf{The ELBO can then be computed simply by plugging each of the derived terms into Equation (9). In these derivations we will occasionally discard some terms that do not depend on the variational parameters, as our purpose of deriving the ELBO is to monitor the convergence of the VB algorithm and those terms are constant across the iterations.

**1st term in (9):**

$$\begin{aligned}E_{q(\tau)}\left[\log p(\tau)\right] &= E_{q(\tau)}\left[(\alpha_0 - 1)\log\tau + (\alpha_0 - 1)\log(1-\tau)\right]\\ &= (\alpha_0 - 1)E_{q(\tau)}\left[\log\tau\right] + (\alpha_0 - 1)E_{q(\tau)}\left[\log(1-\tau)\right]\\ &= (\alpha_0 - 1)\left[\psi(\alpha_\tau) - \psi(\alpha_t + \beta_\tau)\right] + (\alpha_0 - 1)\left[\psi(\beta_\tau) - \psi(\alpha_t + \beta_\tau)\right].\end{aligned}$$

The last line above followed from the known formula for $E_{q(\tau)}\left[\log\tau\right]$ when $q(\tau)$ has the Beta distribution specified in Equation (6).

**2st term in (9):**

$$\begin{aligned}E_{q(\theta)}\left[\log p(\theta)\right] &= E_{q(\theta)}\left[-\frac{\beta_0}{2}\theta^2\right] = -\frac{\beta_0}{2}E_{q(\theta)}\left[\theta^2\right]\\ &= -\frac{\beta_0}{2}\left[Var(\theta) + E_{q(\theta)}(\theta)^2\right]\\ &= -\frac{\beta_0}{2}\left(\beta_2^{-1} + m_2^2\right).\end{aligned}$$

The last line followed directly from the normal distribution (7) for $\theta$.

**3st term in (9):**

$$\begin{aligned}E_{q(\mathbf{z})q(\tau)}\left[\log p(\mathbf{z}|\tau)\right] &= \sum_{n=1}^{N} E_{q(z_n)q(\tau)}[\log p(z_n|\tau)]\\ &= \sum_{n=1}^{N} E_{q(z_n)q(\tau)}[z_{n2}\log\tau + z_{n1}\log(1-\tau)]\\ &= \sum_{n=1}^{N}\left\{E_{q(z_n)}[z_{n2}]E_{q(\tau)}[\log\tau] + E_{q(z_n)}[z_{n1}]E_{q(\tau)}[\log(1-\tau)]\right\}\\ &= \sum_{n=1}^{N}\left\{r_{n2}[\psi(\alpha_\tau) - \psi(\alpha_t + \beta_\tau)] + r_{n1}[\psi(\beta_\tau) - \psi(\alpha_\tau + \beta_\tau)]\right\}.\end{aligned}$$

**4st term in (9):**

$$
\begin{aligned}
E_{q(\mathbf{z})q(\theta)}\left[\log p(\mathbf{x}|\mathbf{z},\theta)\right] &= \sum_{n=1}^{N} E_{q(z_n)q(\theta)}\left[\log p(x_n|z_n\theta)\right] \\
&= \sum_{n=1}^{N} E_{q(z_n)q(\theta)}\left[z_{n1}\log N(x_n|0,1) + z_{n2}\log N(x_n|\theta,1)\right] \\
&= \sum_{n=1}^{N}\left\{E_{q(z_n)}[z_{n1}]\left[-\frac{1}{2}\log(2\pi) - \frac{1}{2}x_n^2\right] + E_{q(z_n)}[z_{n2}]\left[-\frac{1}{2}\log(2\pi) - \frac{1}{2}E_{q(\theta)}[(x_n-\theta)^2]\right]\right\} \\
&= \sum_{n=1}^{N}\left\{r_{n1}\left[-\frac{1}{2}\log(2\pi) - \frac{1}{2}x_n^2\right] + r_{n2}\left[-\frac{1}{2}\log(2\pi) - \frac{1}{2}E_{q(\theta)}[(x_n - m_2 + m_2 - \theta)^2]\right]\right\} \\
&= -\frac{N}{2}\log(2\pi) - \frac{1}{2}\sum_{n=1}^{N} r_{n1}x_n^2 - \frac{1}{2}\sum_{n=1}^{N} r_{n2}\left\{(x_n - m_2)^2 + E_{q(\theta)}[(m_2-\theta)^2]\right\} \\
&= -\frac{N}{2}\log(2\pi) - \frac{1}{2}\sum_{n=1}^{N} r_{n1}x_n^2 - \frac{1}{2}\sum_{n=1}^{N} r_{n2}\left\{(x_n - m_2)^2 + \beta_2^{-1}]\right\}.
\end{aligned}
$$

**5st term in (9):**

$$
\begin{aligned}
E_{q(\mathbf{z})}[\log q(\mathbf{z})] &= \sum_{n=1}^{N} E_{q(z_n)}\left[z_{n1}\log r_{n1} + z_{n2}\log r_{n2}\right] \\
&= \sum_{n=1}^{N} r_{n1}\log r_{n1} + r_{n2}\log r_{n2}.
\end{aligned}
$$

**6st term in (9):**

$$
E_{q(\tau)}\left[\log q(\tau)\right] = \log\frac{\Gamma(\alpha_\tau+\beta_\tau)}{\Gamma(\alpha_\tau)\Gamma(\beta_\tau)} + (\alpha_\tau - 1)\psi(\alpha_\tau) + (\beta_\tau - 1)\psi(\beta_\tau) - (\alpha_\tau + \beta_\tau - 2)\psi(\alpha_\tau+\beta_\tau).
$$

This is just the negative entropy of the $Beta(\alpha_\tau, \beta_\tau)$ distribution (see Wikipedia).

**7st term in (9):**

$$
E_{q(\theta)}[\log q(\theta)] = \frac{1}{2}\log\left(\frac{\beta_2}{2\pi}\right) - \frac{1}{2}.
$$

This is the negative entropy of $N(m_2, \beta_2^{-1})$.

## Problem 2: ELBO for the simple model (2/2)

**(a)** Derive the 4th term $E\_\{q(z)q(\theta)\}\ [\log p(x \mid z, \theta)]$ of the ELBO.

**Hint 1**: $E(XY) = E(X)E(Y)$ if X and Y are assumed indendent (see how this is already used in the derivation of term 3).

**Hint 2**: $E(X - Y) = E(X - a + a - Y)^2$.

**(b)** Implement terms 2, 4, and 7 in the code template given below. Verify that the ELBO increases when you run the VB algorithm.

```
[ ]:    # Template for problem 2(b).
        import numpy as np
        import matplotlib.pyplot as plt

        np.random.seed(123123123)

        # Compute ELBO for the model described in simple_elbo.pdf
        def compute_elbo(alpha_tau, beta_tau, r1, r2, m2, beta2, alpha0, beta0,␣
    ↪x):

        from scipy.special import psi, gammaln # digamma function, logarithm of␣
    ↪gamma function

        # E[log p(tau)]
        term1 = (alpha0 - 1) * (psi(alpha_tau) + psi(beta_tau) - 2 *␣
    ↪psi(alpha_tau + beta_tau))

        # E[log p(theta)]
        # term2 = ?

        ### BEGIN SOLUTION
        term2 = -0.5 * beta0 * (beta2**(-1) + m2**2)
        ### END SOLUTION

        # E[log p(z|tau)]
        N2 = np.sum(r2); N1 = np.sum(r1); N = N1 + N2
        term3 = N2 * psi(alpha_tau) + N1 * psi(beta_tau) - N * psi(alpha_tau +␣
    ↪beta_tau)

        # E[log p(x|z,theta)]
        # term4 = ?

        ### BEGIN SOLUTION
        term4 = -0.5 * np.sum(r1 * x**2) - 0.5 * np.sum(r2 * ((x-m2)**2 +␣
    ↪beta2**(-1)))
        ### END SOLUTION


        # Negative entropy of q(z)
        term5 = np.sum(r1 * np.log(r1)) + np.sum(r2 * np.log(r2))

        # Negative entropy of q(tau)
        term6 = (gammaln(alpha_tau + beta_tau) - gammaln(alpha_tau) -␣
    ↪gammaln(beta_tau)
        + (alpha_tau - 1) * psi(alpha_tau) + (beta_tau - 1) * psi(beta_tau)
        - (alpha_tau + beta_tau - 2) * psi(alpha_tau + beta_tau))
        # Negative entropy of q(theta)
        # term7 = ?

        ### BEGIN SOLUTION
        term7 = 0.5 * np.log(beta2)
        ### END SOLUTION

        elbo = term1 + term2 + term3 + term4 - term5 - term6 - term7

        return elbo


        # Simulate data
        theta_true = 4
        tau_true = 0.3
        n_samples = 10000
        z = (np.random.rand(n_samples) < tau_true)  # True with probability␣
↪tau_true
        x = np.random.randn(n_samples) + z * theta_true

        # Parameters of the prior distributions.
        alpha0 = 0.5
        beta0 = 0.2

        n_iter = 15 # The number of iterations
        elbo_array = np.zeros(n_iter) # To track the elbo

        # Some initial value for the things that will be updated
        E_log_tau = -0.7   # E(log(tau))
        E_log_tau_c = -0.7  # E(log(1-tau))
        E_log_var = 4 * np.ones(n_samples)  # E((x_n-theta)^2)
        r2 = 0.5 * np.ones(n_samples)  # Responsibilities of the second cluster.

        for i in range(n_iter):

        # Updated of responsibilites, factor q(z)
        log_rho1 = E_log_tau_c - 0.5 * np.log(2 * np.pi) - 0.5 * (x ** 2)
        log_rho2 = E_log_tau - 0.5 * np.log(2 * np.pi) - 0.5 * E_log_var
        max_log_rho = np.maximum(log_rho1, log_rho2)  # Normalize to avoid␣
↪numerical problems when exponentiating.
        rho1 = np.exp(log_rho1 - max_log_rho)
        rho2 = np.exp(log_rho2 - max_log_rho)
        r2 = rho2 / (rho1 + rho2)
        r1 = 1 - r2

        N1 = np.sum(r1)
        N2 = np.sum(r2)
        # Update of factor q(tau)
        from scipy.special import psi # digamma function
        E_log_tau = psi(N2 + alpha0) - psi(N1 + N2 + 2*alpha0)
        E_log_tau_c = psi(N1 + alpha0) - psi(N1 + N2 + 2*alpha0)

        # Update of factor q(theta)
        x2_avg = 1 / N2 * np.sum(r2 * x)
        beta_2 = beta0 + N2
        m2 = 1 / beta_2 * N2 * x2_avg
        E_log_var = (x - m2) ** 2 + 1 / beta_2

        # Keep track of the current estimates
        tau_est = (N2 + alpha0) / (N1 + N2 + 2*alpha0)
        theta_est = m2

        # Compute ELBO
        alpha_tau = N2 + alpha0
        beta_tau = N1 + alpha0
        elbo_array[i] = compute_elbo(alpha_tau, beta_tau, r1, r2, m2, beta_2,␣
 ↪alpha0, beta0, x)

        # Plot ELBO as a function of iteration
        plt.plot(np.arange(n_iter) + 1, elbo_array)
        plt.xticks(np.arange(n_iter) + 1)
        plt.xlabel("iteration")
        plt.title("ELBO")
        plt.show()
```

## Problem 3: Bayes factors

Suppose we have two bags, each containing a large number of black and white marbles. To learn about the contents of the bags, we have done 5 draws from each bag. After each draw, the marble drawn has been returned to the bag. The draws from the first bag are as follows ( B, W, W, B, B ) and the draws from the second bag are ( B, B, B, B, W ) , where B corresponds to a Black marble and W to a White marble.

Consider two models:

- $M_1$ : the proportions of marbles are the same in the two bags
- $M_2$ : the proportions of marbles are different in the two bags.

**(a)** Write out the two models explicitly. Assuming that a priori all proportions are equally probable, compute the Bayes factor in favor of $M_1$.

**(b)** The same as (a), but now the first set of draws contains 300 black and 200 white draws, and second set of draws 250 black and 250 white draws.

**Hint**: Beta distribution is the conjugate prior for the Binomial/Bernoulli likelihood, and a uniform proportion corresponds to the $Beta(1,1)$ distribution.

**Solution**

We can express the model $M_1$ as

$$\begin{aligned} x_i &\sim Binomial(n,p) \quad i = 1,2 \\ p &\sim Beta(1,1) \end{aligned}$$

and model $M_2$ as

$$\begin{aligned} x_i &\sim Binomial(n,p_i) \quad i = 1,2 \\ p_i &\sim Beta(1,1) \end{aligned}$$

where $x_1$ and $x_2$ denote the number of white marbles drawn from the first and second bag, respectively, and $n$ is number of marbles drawn from each bag.

For computing the Bayes factor we need the marginal likelihoods for each model, derived as follows. For model $M_1$ we get

$$\begin{aligned} p(x|M_1) &= \int_0^1 Binomial(x_1|n,p)Binomial(x_2|n,p)Beta(p|1,1)dp \\ &= \binom{n}{x_1}\binom{n}{x_2}\int_0^1 p^{x_1}(1-p)^{n-x_1}p^{x_2}(1-p)^{n-x_2}dp \\ &= \binom{n}{x_1}\binom{n}{x_2}\int_0^1 p^{x_1+x_2+1-1}(1-p)^{2n-x_1-x_2+1-1}dp \\ &= \binom{n}{x_1}\binom{n}{x_2}B(x_1+x_2+1, 2n-x_1-x_2+1). \end{aligned}$$

where $B(\cdot,\cdot)$ denotes the beta function. In model $M_2$ we get the marginal likelihood as

$$\begin{aligned} p(x|M_2) &= \int_0^1\int_0^1 Binomial(x_1|n,p_1)Binomial(x_2|n,p_2)Beta(p_1|1,1)Beta(p_2|1,1)dp_1dp_2 \\ &= \binom{n}{x_1}\binom{n}{x_2}\int_0^1\int_0^1 p_1^{x_1}(1-p_1)^{n-x_1}p_2^{x_2}(1-p_2)^{n-x_2}dp_1dp_2 \\ &= \binom{n}{x_1}\binom{n}{x_2}B(x_1+1, n-x_1+1)B(x_2+1, n-x_2+1). \end{aligned}$$

So the Bayes factor is given as

$$K = \frac{p(x|M_1)}{p(x|M_2)} = \frac{B(x_1+x_2+1, 2n-x_1-x_2+1)}{B(x_1+1, n-x_1+1)B(x_2+1, n-x_2+1)}.$$

In (a) the values were $n = 5$, $x_1 = 2$ and $x_2 = 1$, so the Bayes factor is $K \approx 1.3636$. In (b) $n = 500$, $x_1 = 200$ and $x_2 = 250$, so $K \approx 0.0816$. With smaller draws from the bags we do not have evidence to believe that the bags have different proportions of marbles, while when drawing a large amount marbles from each bag the situation changes.

## Problem 4: Model selection for GMM with BIC and cross validation

In many machine learning applications model selection is crucial. In this exercise, you will practice two common approaches for model selection:

- Bayesian Information Criterion (BIC) (as an approximation to 'Bayesian model selection') and
- Cross-Validation (as a representative for a predictive model selection criterion).

You are given a data set (1000 samples of dimension 2) contained in the file data.pickle, which has been sampled from a Gaussian Mixture Model (GMM) using three classes(the true class labels are given for your convenience, but they should not be used in learning the model).

In the given code template below, the data will be divided into training and test sets.

**(a)** Complete the functions 'compute_bic' and 'cross_validate'. Use both criteria to select the number of components in the GMM using the training data. Plot the both the BIC and the validation log-likelihoods as a function of the number of components, as well as the data with the best model. Do both methods find a model with three components as the most likely?

**(b)** Use the selected models to evaluate the test set log-likelihood.

**(c)** Explain briefly the pros and cons of the two approaches and comment which approach you would consider better and why.

**Hint**: What is the total number of parameters needed to specify the component means and covariance matrices, and the mixture weights? You will need this number to compute BIC.

[ ]:
```
        # This Starter code for problem 4. The solution template is in the cell␣
↪below.
        # Tools for learning Gaussian mixture models; adapted from the␣
↪BRMLtoolkit by David Barber.
        import numpy as np
        import numpy.matlib as matlib
        import numpy.linalg as LA
        #import scipy.misc
        from scipy.special import logsumexp

        def condp(X):
        return X / np.sum(X, axis=0)

        def condexp(logp):
        pmax = np.max(logp, axis=0)
        P = logp.shape[0]
        return condp(np.exp(logp - np.tile(pmax, (P, 1))))

        def GMMlogp(X, H, P, m, S):
        D, N = X.shape  # dimension and number of points

        logp = np.zeros((N, H))
        for i in range(H):
```

```
invSi = LA.inv(S[i,:,:])
sign, logdetSi = LA.slogdet(2 * np.pi * S[i,:,:])

for n in range(N):
v = X[:,n] - m[:,i]
logp[n,i] = -0.5 * (v @ invSi @ v) - 0.5 * logdetSi + np.log(P[i])

return logp



# Log Likelihood of data X under a Gaussian Mixture Model
#
# X : each column of X is a datapoint.
# P : mixture coefficients
# m : means
# S : covariances
#
# Returns: A list containing the log likelihood for each data point in X
def GMMloglik(X, P, m, S):
N = X.shape[1]
H = m.shape[1]

logp = GMMlogp(X, H, P, m, S)
logl = [logsumexp(a=logp[n,:], b=np.ones(H)) for n in range(N)]

return logl

# Fit a mixture of Gaussian to the data X using EM
#
# X : each column of X is a datapoint.
# H : number of components of the mixture.
# n_iter : number of EM iterations
#
# Returns: (P, m, S, loglik, phgn)
# P : learned mixture coefficients
# m : learned means
# S : learned covariances
# loglik : log likelihood of the learned model
# phgn : mixture assignment probabilities
def GMMem(X, H, n_iter):
D, N = X.shape  # dimension and number of points

# initialise the centres to random datapoints
r = np.random.permutation(N)
m = X[:, r[:H]]
```

```
# initialise the variances to be large
s2 = np.mean(np.diag(np.cov(X)))
S = matlib.tile(s2 * np.eye(D), [H, 1, 1])

# intialise the component probilities to be uniform
P = np.ones(H) / H

for emloop in range(n_iter):
# E-step:
logpold = GMMlogp(X, H, P, m, S)

phgn = condexp(logpold.T)  # responsibilities
pngh = condp(phgn.T)       # membership

# M-step:
for i in range(H):   # now get the new parameters for each component
tmp = (X - np.tile(m[:,i:i+1], N)) * np.tile(np.sqrt(pngh[:,i]), (D,1))
Scand = np.dot(tmp, tmp.T)

if LA.det(Scand) > 0.0001:   # don't accept too low determinant
S[i,:,:] = Scand

m = np.dot(X, pngh)
P = np.sum(phgn, axis=1) / N

logl = np.sum(logsumexp(a=logpold[n,:], b=np.ones(H)) for n in range(N))

return P, m, S, logl, phgn
```

[ ]:
```
# Template for problem 4
import matplotlib.pyplot as plt
import pickle

np.random.seed(0)
totalComponents = 5  # max number of mixture components
criterion_flag = 0    # 0: cross validation, 1: BIC

def compute_bic(Xtrain):
BICs = []
for H in range(1, totalComponents+1):    # number of mixture components
print("H: {}".format(H))

P, m, S, loglik, phgn = GMMem(Xtrain, H, 100)  # fit to data

# numParams = ?     # number of parameters in the model
# BIC = ?           # BIC for the model
```

```
        ### BEGIN SOLUTION
        numParams = H * D*(D+1)/2 + H*D + (H-1)
        BIC = -2*loglik + numParams * np.log(Xtrain.shape[1])
        ### END SOLUTION

        BICs.append(BIC)
        return BICs

        def cross_validate(Xtrain):
        foldCount = 5    # number of folds

        loglik = np.zeros((totalComponents, foldCount))

        Nlearning = Xtrain.shape[1]
        order = np.random.permutation(Nlearning)    # to randomize the sample␣
  ↪order

        for H in range(1, totalComponents+1):     # number of mixture components
        print("H: {}".format(H))

        for fold in range(foldCount):    # K-fold cross validation (K=5)
        ind = fold * int(Nlearning/foldCount) + np.arange(int(Nlearning/
  ↪foldCount))
        val_indices = order[ind]

        training_indices = np.setdiff1d(np.arange(Nlearning), val_indices);

        X_train = Xtrain[:,training_indices]  # cv training data
        X_val   = Xtrain[:,val_indices]       # cv validation data

        # train model
        P1, m1, S1, loglik1, phgn1 = GMMem(X_train, H, 100)   # fit model

        # Predict using the cv trained model
        # logl1 = ?
        # loglik[H-1,fold] = ?

        ### BEGIN SOLUTION
        logl1 = GMMloglik(X_val, P1, m1, S1)
        loglik[H-1,fold] = np.sum(logl1)
        ### END SOLUTION
        return loglik


        # load data
        with open("/coursedata/data.pickle", "rb") as f:
        X, labels = pickle.load(f)
```

```
       D, N = X.shape    # dimension and number of data points

       ratio = 0.75
       train_ind = np.random.choice(N, int(ratio * N), replace=False)   #␣
↪training data index
       test_ind = np.setdiff1d(np.arange(N), train_ind)                 # test␣
↪data index

       Xtrain = X[:,train_ind]            # training data
       Xtrain_labels = labels[train_ind]  # training data labels

       Xtest = X[:,test_ind]            # test data
       Xtest_labels = labels[test_ind]  # test data labels

       # plot training and test data
       def plot_data():
       for i in sorted(set(Xtrain_labels)):
       X_comp = Xtrain[:, Xtrain_labels == i]
       plt.plot(X_comp[0], X_comp[1], '.' + 'brgmcyk'[i-1], markersize=6)

       plt.plot(Xtest[0], Xtest[1], 'kd', markersize=4, markeredgewidth=0.5,␣
↪markerfacecolor="None")

       plot_data()
       plt.title('training data, test data (in black)')
       plt.show()



       # ***** Use BIC to select the number of components
       # (Only this part differs from the second template, where cross␣
↪validation is used instead)

       if criterion_flag:
       print('computing BIC')
       scores = compute_bic(Xtrain)
       ylabel = "(BIC)"
       elif not criterion_flag:
       print('computing cross validation score')
       scores = cross_validate(Xtrain)
       ylabel = "(Cross Validation)"

       # plot the BIC curve
       plt.bar(np.arange(1, totalComponents+1), np.mean(scores, axis=1))
       #plt.yscale("log", nonposy="clip")
       plt.xlabel('Number of Mixture Components')
```

```python
plt.ylabel(ylabel)
plt.title('Model Selection' + ylabel )
plt.show()

# select the number of mixture components which minimizes the BIC
h = np.argmax(np.mean(scores, axis=1)) + 1



# ***** TRAIN

# Now train full model with selected number of mixture components
P, m, S, loglik, phgn = GMMem(Xtrain, h, 100)  # fit to data

# Predict using the full trained model (Use GMMem.GMMloglik)
# logl = ?
# print('Test Data Likelihood = {0:f}'.format(?))

### BEGIN SOLUTION
logl = GMMloglik(Xtest, P, m, S)
print('Test Data Likelihood = {0:f}'.format(np.sum(logl)))
### END SOLUTION

# Plot the best GMM model
plot_data()

for i in range(h):
dV, E = LA.eig(S[i,:,:])

theta = np.arange(0, 2*np.pi, 0.1)
p = np.sqrt(dV.reshape(D,1)) * [np.cos(theta), np.sin(theta)]
x = (E @ p) + np.tile(m[:,i:i+1], (1, len(theta)))

plt.plot(x[0], x[1], 'r-', linewidth=2)

plt.title('training data, test data (in black)')
plt.show()
```