Assigment 2. Advanced machine learning.

Author:
Rodrigo Castellano Ontiveros
roco@kth.se

Question 2.1.1: Yes

Question 2.1.2: No

Question 2.1.3: Yes

Question 2.1.4: No

Question 2.1.5: No

Question 2.1.6: No

Question 2.2.7: Implement a dynamic programming algorithm that, for a given T,Θ and β, computes $p(\beta|T,\Theta)$.

We have a rooted binary tree T, where the vertex set is denoted by $V(T)$ and the leaf set by $L(T)$. For a vertex v belonging to the vertex set V, there is an associated random variable X_v. The CPD of a vertex v is defined as $\Theta_v = p(X_v|x_{pa(v)})$. Θ_v is a categorical distribution and $pa(v)$ is the parents of v.

Moreover, for a leaf $l \in L(T)$, we also define $\beta = \{xl : l \in L(T)\}$, which is an assignment of values to the set $L(T)$.

The expression $p(\beta|T,\Theta)$ is defined as follows [1]:

$$p(\beta|T,\Theta) = \sum_i s(r,i)p(X_r = i) \tag{1}$$

where r denotes the root node, X_r is its associated random variable that takes values in $[K]$, and $s(r,i)$ is the probability of all the observations below the node r, conditioned by the random variable X_r having the value i.

This problem is solved by recursion on the function $s(v,i)$. In the rooted binary tree T, for $v \in V(T)$ with children $u, w \in V(T)$,

$$s(v,i) = p(X_{o\cap\downarrow v}|X_v = i) = p(X_{o\cap\downarrow w}|X_v = i)p(X_{o\cap\downarrow u}|X_v = i) =$$
$$\left(\sum_{j\in 1,\dots,K} p(X_w = j|X_v = i)s(w,j)\right)\left(\sum_{j\in 1,\dots,K} p(X_u = j|X_v = i)s(u,j)\right)$$

For a leaf $l \in L(T)$,

$$s(l,i) = \begin{cases} 0 & \text{if } \mathrm{x}_l \neq i \\ 1 & \text{if } \mathrm{x}_l = i \end{cases} \tag{2}$$

The problem can be solved by starting at the root as in expression (1) and going through every vertex until the leafs are reached. Another way to solve it is to start from the leafs with expression (2) and go until the root node. The procedure is shown in [1].

Question 2.2.8: Report $p(\beta|T,\Theta)$ for each given data, separately. You should report 15 values in total; 5 for small tree, 5 for medium tree and 5 for large tree.

The results are shown in table (1). The results seem to be correct when it comes to comparing probabilities and the trees' sizes. For the small tree, the order of the likelihoods is 10^{-2}, for the medium tree it is 10^{-18}, and for the large tree 10^{-69}, approximately. In general, the larger the tree is, the smaller the probability $p(\beta|T,\Theta)$ is for a given β.

For the large tree, there are so many possible different β that the probability of one of them is very low. We can see in table (1) that the order of the sample 4 is 10^5 times higher than in sample 5, but still the order is 10^{-67}.

Tree and Sample	Likelihood
Small Tree, 1	0.009786075668602368
Small Tree, 2	0.015371111945909397
Small Tree, 3	0.02429470256988136
Small Tree, 4	0.005921848333806081
Small Tree, 5	0.016186321212555956
Medium Tree, 1	1.7611947348905713e-18
Medium Tree, 2	2.996933026124685e-18
Medium Tree, 3	2.891411201505415e-18
Medium Tree, 4	4.6788419411270006e-18
Medium Tree, 5	5.664006737201378e-18
Large Tree, 1	3.63097513075208e-69
Large Tree, 2	3.9405421986921234e-67
Large Tree, 3	5.549061147187144e-67
Large Tree, 4	9.89990102807915e-67
Large Tree, 5	3.11420969368965e-72

Table 1: Results for each of the samples.

Question 2.3.9: Implement the VI algorithm for the variational distribution in Equation (10.24) in Bishop.

All the steps are shown in [2]. The VI algorithm tries to find the the posterior $p(\mu,\tau)$ for a given dataset $D = \{x_1, ..., x_N\}$ with each observed value x independently drawn from a Gaussian distribution. We try to find a variational approximation of the posterior by using mean field theory. The assumption is that $q(\mu,\tau) = q_\mu(\mu)q_\tau(\tau)$.

The priors are defined as:

$$\begin{aligned} p(\mu \mid \tau) &= \mathcal{N}\left(\mu \mid \mu_0, (\lambda_0\tau)^{-1}\right) \\ p(\tau) &= \text{Gam}\left(\tau \mid a_0, b_0\right) \end{aligned} \tag{3}$$

For an observed value x_n,

$$p(x_n \mid \mu, \tau) = \left(\frac{\tau}{2\pi}\right)^{1/2} \exp\left\{-\frac{\tau}{2}(x_n - \mu)^2\right\} \tag{4}$$

For the dataset,

$$p(\mathcal{D} \mid \mu, \tau) = \left(\frac{\tau}{2\pi}\right)^{N/2} \exp\left\{-\frac{\tau}{2}\sum_{n=1}^{N}(x_n - \mu)^2\right\} \tag{5}$$

To compute each optimal factor, the expression applied is

$$\ln q_j^\star(\mathbf{Z}_j) = \mathbb{E}_{i\neq j}[\ln p(\mathbf{X}, \mathbf{Z})] + \text{ const.} \tag{6}$$

Therefore, we can apply this expression to each of the factors. The result for each of them is

$$\begin{aligned}
\ln q_\mu^\star(\mu) &= \mathbb{E}_\tau[\ln p(\mathcal{D} \mid \mu, \tau) + \ln p(\mu \mid \tau)] + \text{ const} \\
&= -\frac{\mathbb{E}[\tau]}{2}\left\{\lambda_0(\mu - \mu_0)^2 + \sum_{n=1}^{N}(x_n - \mu)^2\right\} + \text{ const.}
\end{aligned}$$

$$\begin{aligned}
\ln q_\tau^\star(\tau) =& \mathbb{E}_\mu[\ln p(\mathcal{D} \mid \mu, \tau) + \ln p(\mu \mid \tau)] + \ln p(\tau) + \text{ const} \\
&= (a_0 - 1)\ln\tau - b_0\tau + \frac{N}{2}\ln\tau \\
&\quad - \frac{\tau}{2}\mathbb{E}_\mu\left[\sum_{n=1}^{N}(x_n - \mu)^2 + \lambda_0(\mu - \mu_0)^2\right] + \text{ const}
\end{aligned} \tag{7}$$

Therefore, by comparing with normal and Gaussian distribution, $q_\mu(\mu) \sim \mathcal{N}\left(\mu \mid \mu_N, (\lambda_N)^{-1}\right)$ and $q_\tau(\tau) \sim Gam(\tau \mid a_N, b_N)$.

To implement the EM algorithm we need to set initial values for $\mu_0, \lambda_0, a_0, b_0$. A initial guess can be done for $\mathbb{E}[\tau] = \frac{a_N}{b_N}$, where$\frac{a_N}{b_N}$ is the mean of the Gamma distribution. We express the moments as $\mathbb{E}[\mu] = \mu_N$ and $\mathbb{E}[\mu^2] = var(\mu) + \mathbb{E}[\mu]^2 = \frac{1}{\lambda_N} + \mu_N^2$.

Then, the following parameters are updated:

$$\begin{aligned}
\mu_N &= \frac{\lambda_0\mu_0 + N\bar{x}}{\lambda_0 + N} \\
\lambda_N &= (\lambda_0 + N)\,\mathbb{E}[\tau] \\
a_N &= a_0 + \frac{N}{2} \\
b_N &= b_0 + \frac{1}{2}\mathbb{E}_\mu\left[\sum_{n=1}^{N}(x_n - \mu)^2 + \lambda_0(\mu - \mu_0)^2\right] \\
&= b_0 + \frac{1}{2}\sum_{n=1}^{N}\left(x_n^2 + \mathbb{E}[\mu^2] - 2x_n\mathbb{E}[\mu] + \lambda_0(\mathbb{E}[\mu^2] + \mu_0^2 - 2\mu_0\mathbb{E}[\mu])\right)
\end{aligned} \tag{8}$$

We can iterate until a convergence criterion is satisfied.

Question 2.3.10: What is the exact posterior?

The exact posterior can be found by using a Normal-Gamma conjugate prior (NG). The main difference with the variational approximation is that now we don't apply the factorization assumption, instead

$$p(\mu, \tau) = p(\mu \mid \tau)p(\tau) \tag{9}$$

Therefore, by applying Bayes' rule,

$$\begin{aligned} p(\mu, \tau \mid D) &= \frac{p(D \mid \mu, \tau)p(\mu, \tau)}{p(D} \propto p(D) \mid \mu, \tau)p(\mu \mid \tau)p(\tau) \\ &\propto p(D \mid \mu, \tau)NG(\mu, \tau \mid \mu_0, \lambda_0, a_o, b_0) \\ &\propto \mathcal{N}\left(\mu \mid \mu_n, ((\lambda + N)\tau)^{-1}\right) \times Ga\left(\tau \mid a_0 + N/2, b_n\right) \end{aligned} \tag{10}$$

The final result is

$$\begin{aligned} p(\mu, \tau \mid D) &= NG\left(\mu, \tau \mid \mu_N, \lambda_N, a_N, b_N\right) \\ \mu_N &= \frac{\lambda_0 \mu_0 + N\bar{x}}{\lambda_0 + N} \\ \lambda_N &= \lambda_0 + N \\ a_N &= a_0 + N/2 \\ b_N &= b_0 + \frac{1}{2}\sum_{i=1}^{N}\left(x_i - \bar{x}\right)^2 + \frac{\lambda_0 N\left(\bar{x} - \mu_0\right)^2}{2\left(\lambda_0 + N\right)} \end{aligned} \tag{11}$$

The information can be found in [3], where the steps are more detailed.

Question 2.3.11: Compare the inferred variational distribution with the exact posterior. Run the inference on data points drawn from i.i.d. Gaussians. Do this for three interesting cases and visualize the results. Describe the differences.

Different results are shown. In figure (4), it is clear that the larger the number of samples is, the better the approximation becomes. For 100 samples, the variational approximation is already very similar to the exact posterior. In figure (8), the main difference is the shapes of the variational distributions. The change in parameters has been to set them all to 1 instead of 0. A greater difference in parameters can be tried, or a change in individual parameters. This could be done in a further analysis.

When it comes to the parameters of the Gaussian from which the points are drawn, figures (12) and (16) show the differences. For figure (12), we can see that the range of the xlabel is what mostly changes for $\mu = 20, 100$. For τ, as expected, the distributions become much smaller as τ decreases.

Other parameters could have been analyzed, such as the convergence criterion.

I believe the results are reasonable and good enough, which means that the variational approximation is very close to the exact posterior. The success strongly depends on the number of samples, as shown in the figures mentioned above. This is clear because in this case because when there is more data, the estimations are better. Other parameters, such as the initialization parameters, can change the shape of the approximation distribution. It is important to try with different initializations.

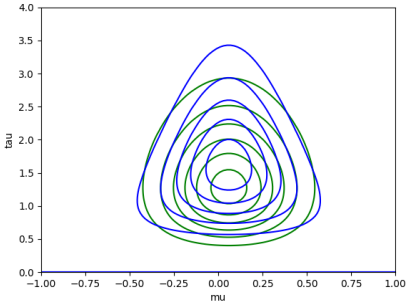

Figure 1: 10 samples.

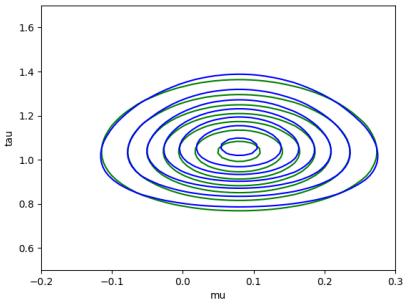

Figure 2: 100 samples.

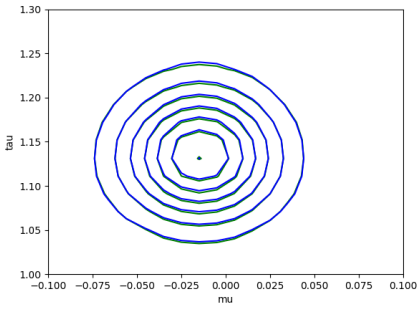

Figure 3: 100 samples.

Figure 4: Initial parameters set to 0, $(\mu, \tau) = (0, 1)$. Diferent number of samples.

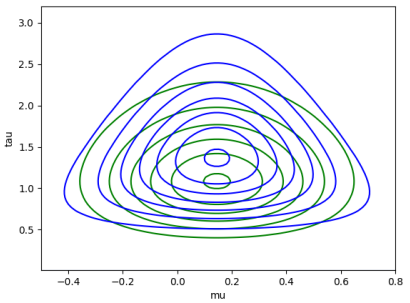

Figure 5: 10 samples.

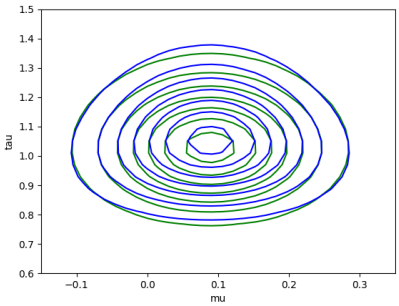

Figure 6: 100 samples.

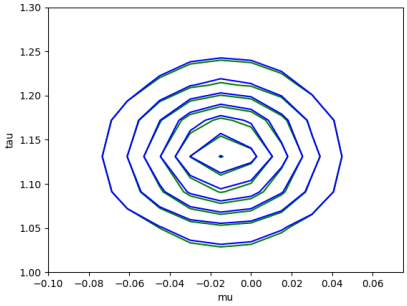

Figure 7: 100 samples.

Figure 8: Initial parameters set to 1, $(\mu, \tau) = (0, 1)$. Diferent number of samples.

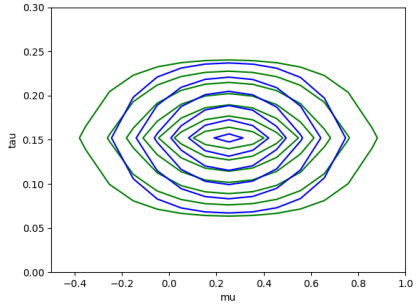

Figure 9: $\mu = 0$.

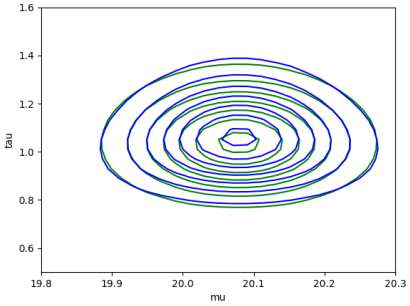

Figure 10: $\mu = 20$.

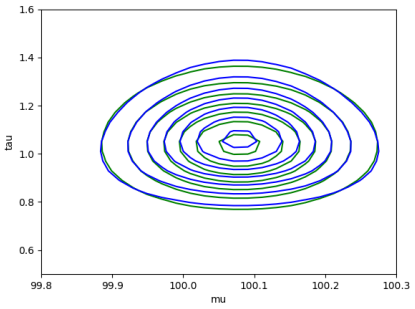

Figure 11: $\mu = 100$.

Figure 12: Initial parameters set to 0, 100 samples. Different values of μ. $\tau = 1$.

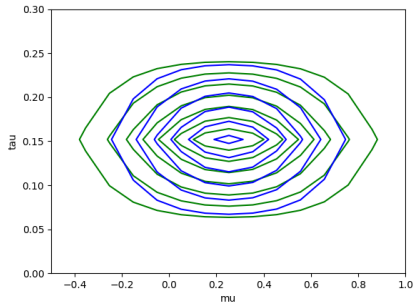

Figure 13: $\tau = 0.1$.

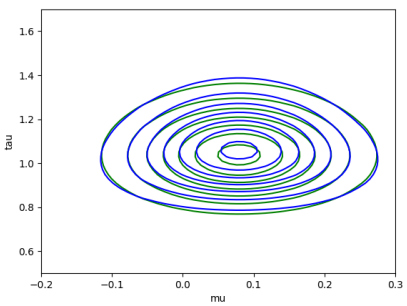

Figure 14: $\tau = 1$.

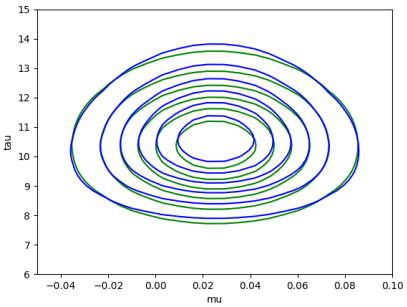

Figure 15: $\tau = 10$.

Figure 16: Initial parameters set to 0, 100 samples. Different values of τ, $\mu = 0$.

Question 2.4.12: Derive an EM algorithm for the model.

The EM algorithm tries to find the maximum likelihood solution for probabilistic models with latent variables [2]. As shown in [2], for hidden variables Z, parameters Θ and observed variables X, it maximizes

$$p(X \mid \theta) = \sum_{Z} p(X, Z \mid \theta) \tag{12}$$

In this case, the set of observed variabels is S and X, and the hidden variables is Z. We optimize the complete log-likelihood because it is easier to work with. Since the variable Z is unknown, the expected value of the complete log-likelihood is computed (with regard to Z). To see the parameters of each variable and their dependencies, we have to look at the graphical model. Therefore,

$$P\left(X, S, Z \mid \theta\right) = p\left(Z \mid \theta\right) p\left(X \mid Z, \theta\right) p\left(S \mid Z, \theta\right) \tag{13}$$

With

$$\begin{aligned} p(Z \mid \pi) &= \prod_{n=1}^{N} \prod_{k=1}^{K} \pi_k^{z_{nk}} \\ p(X \mid Z, \mu, \tau) &= \prod_{n=1}^{N} \prod_{k=1}^{K} \mathcal{N}(x_n \mid \mu_k, \tau_k^{-1} I)^{z_{nk}} \\ p(S \mid Z, \lambda) &= \prod_{n=1}^{N} \prod_{k=1}^{K} P(s_n \mid \lambda_k)^{z_{nk}} \end{aligned} \tag{14}$$

1. Initialize $\pi, \mu, \Sigma = \tau^{-1} I$ and evaluate the log-likelihood.

2. In the E-step of the EM algorithm, the responsabilities (defined below) are calculated to later update parameters in the M-step.

$$\gamma\left(z_{nk}\right) = \frac{\pi_k \mathcal{N}(x_n \mid \mu_k, \Sigma_k) P(\lambda_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}\left(x_n \mid \mu_j, \Sigma_j\right) P(\lambda_j)} \tag{15}$$

3. In the M-step, for each cluster, m_k has to be calculated and the update the parameters π, μ, Σ, γ.

$$\begin{aligned} \mu_k^{\text{new}} &= \frac{1}{N_k} \sum_{n=1}^{N} \gamma\left(z_{nk}\right) x_n \\ \Sigma_k^{\text{new}} &= \frac{1}{N_k} \sum_{n=1}^{N} \gamma\left(z_{nk}\right) \left(x_n - \mu_k^{\text{new}}\right) \left(x_n - \mu_k^{\text{new}}\right)^{\mathrm{T}} \\ \pi_k^{\text{new}} &= \frac{N_k}{N} \end{aligned} \tag{16}$$

where

$$N_k = \sum_{n=1}^{N} \gamma\left(z_{nk}\right) \tag{17}$$

4. Calculate the log-likelihood and find out if convergence criterion is satisfied.

$$\ln p(X, S \mid \mu, \Sigma, \pi) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}\left(x_n \mid \mu_k, \Sigma_k\right) \right\} \tag{18}$$

Question 2.4.13: Implement your EM algorithm.

The implemented EM algorithm can be found in the appendix.

Question 2.4.14: Apply it to the data provided, give an account of the success, and provide visualizations for a couple of examples. Repeat it for a few different choices of K.

In figure 22 the results are shown. Depending on the cluster, the approximations are good or not. For $K = 1$ it is clear that only one Gaussian does not capture the structure of the data and it's too general. Plenty of points are no accurately described by the Gaussian. For $K = 2$, there approximation seems to be not sufficiently good, given that there is a clear cluster in the center, and none of the Gaussians reflect it properly. For $K = 3$, the approximation is really good, I would say it is the ideal number of clusters. It properly describes every cluster and almost all the points are described accurately by a Gaussians. In each Gaussian the Poisson count is represented by the thickness of the lines. For higher K, such as 4 or 5, we can see that there are too many Gaussians. In this case, there is an excessive number them in the cluster of the center.

By simple visualization, we can see that $K = 3$ is the best choice. This method is successful because the data is divided in different normal distributions. Other methods could have been tried, such as KNN, but it does not give us ellipsoid shapes. The requirement for this unsupervised learning model is to set the number of clusters, but in this case the dataset allows to see how many are needed, with no further need to do more computations.

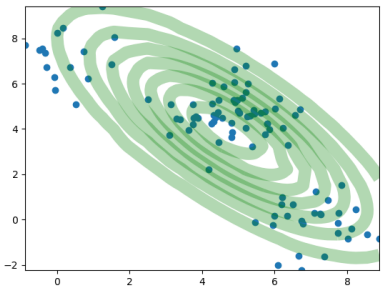

Figure 17: $K = 1$.

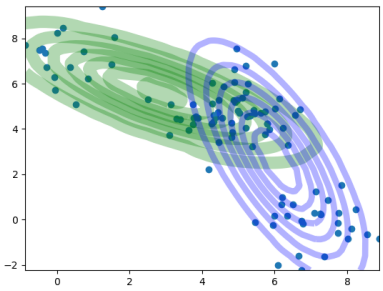

Figure 18: $K = 2$.

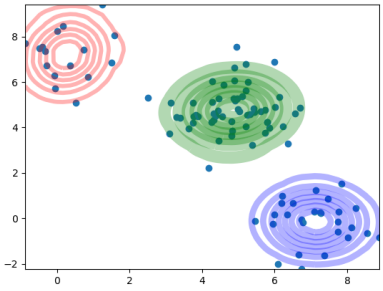

Figure 19: $K = 3$.

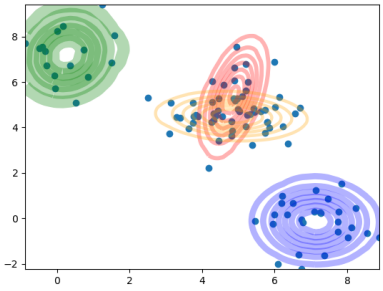

Figure 20: $K = 4$.

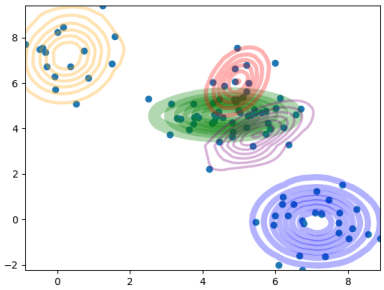

Figure 21: $K = 5$.

Figure 22: Visualization of the clusters.

References

[1] *Course Advanced machine learning lectures*. 2021.

[2] Christopher M. Bishop. *Pattern recognition and Machine Learning.* Springer., 2006.

[3] Kevin P. Murphy. Conjugate bayesian analysis of the gaussian distribution. `https://www.cs.ubc.ca/~murphyk/Papers/bayesGauss.pdf`.

A Apendix A: Code

There are three scripts. The first one corresponds to task 2.2, the second one corresponds to task 2.3 and the last one to task 2.4.

```python
import numpy as np
from Tree import Tree
from Tree import Node


def calculate_likelihood(tree_topology, theta, beta):
    """
    This function calculates the likelihood of a sample of leaves.
    :param: tree_topology: A tree topology. Type: numpy array. Dimensions: (num_nodes, )
    :param: theta: CPD of the tree. Type: list of numpy arrays. Dimensions (approximately): (num_nodes,
K, K)
    :param: beta: A list of node assignments. Type: numpy array. Dimensions: (num_nodes, )
                Note: Inner nodes are assigned to np.nan. The leaves have values in [K]
    :return: likelihood: The likelihood of beta. Type: float.
    """

    num_nodes = len(tree_topology)
    s = np.zeros((num_nodes,k))
    s[:] = np.nan

    # Go through every node, from leaf to root
    for i in range(num_nodes-1,-1,-1):
        # Calculate children
        children = []
        for l in range (len(tree_topology)):
            if tree_topology[l] == i:
                children.append(l)
        # Check if it's a leaf
        if np.isnan(beta[i]) == True:
            # Go through all the possible values in k that the node can take
            for m in range(k):
                # Go through every child and sum its s
                mul = 1
                for child in children:
                    sum = 0
                    for j in range(k):
                        sum += s[child,j]*theta[child][m][j]
                    mul = mul*sum
                s[i,m] = mul
        else:
            # If it's a leaf assign it to s
            for j in range(k):
                if j == beta[i]:
                    s[i,j] = 1
                else:
                    s[i,j] = 0
    # Calculate the likelihood
    likelihood = 0
    for i in range(k):
        likelihood += s[0,i]*theta[0][i]
    return likelihood


def main():

    # filename = "data/q2_2_small_tree.pkl"
    filename = "data/q2_2_large_tree.pkl"
    # filename = "data/q2_2_medium_tree.pkl"
    print("filename: ", filename)

    t = Tree()
    t.load_tree(filename)
    # t.print()

    print("K of the tree: ", t.k, "\talphabet: ", np.arange(t.k))
    print('\ttree topo', t.get_topology_array())
    # print('theta: ', t.get_theta_array())
    global k
    k = t.k

    print("\n2. Calculate likelihood of each FILTERED sample\n")
    # These filtered samples already available in the tree object.
    # Alternatively, if you want, you can load them from corresponding .txt or .npy files

    for sample_idx in range(t.num_samples):
        beta = t.filtered_samples[sample_idx]
        print("\nSample: ", sample_idx, "\tBeta: ", beta)
        sample_likelihood = calculate_likelihood(t.get_topology_array(), t.get_theta_array(), beta)
        print("\tLikelihood: ", sample_likelihood)



if __name__ == "__main__":
    main()
```

```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from  scipy.stats import gamma
from  scipy.stats import norm

def iterations(a_0, b_0, lambda_0, mu_0, X, X_mean):
    # Initial guess
    E_tau = 1
    mu_N = 0
    old_E_tau = 1-E_tau
    # Update parameters
    while abs(old_E_tau-E_tau)>10**(-6):

        a_N = a_0 + N/2
        mu_N = (lambda_0*mu_0 + N*X_mean)/(lambda_0 + N)
        lambda_N = (lambda_0 + N)*E_tau

        E_mu = mu_N
        E_mu2 = 1/lambda_N + E_mu**2
        expr_1 = np.sum(X*X) -2*np.sum(X)*mu_N + N*E_mu2
        expr_2 = lambda_0*( E_mu2 + mu_0**2-2*mu_0*E_mu)
        b_N = b_0 + 0.5*(expr_1 + expr_2)

        old_E_tau = E_tau
        # Update tau
        E_tau = a_N/b_N

    return a_N, b_N, lambda_N, mu_N

def exact_posterior(mu, tau, a_0, b_0, lambda_0, mu_0, X, X_mean):
    a_N = a_0 + N/2
    mu_N = (lambda_0*mu_0 + N*X_mean)/(lambda_0 + N)
    lambda_N = (lambda_0 + N)

    expr_1 = np.sum(X*X) -2*np.sum(X)*X_mean + N*X_mean**2
    expr_2 = lambda_0*N*( X_mean**2 + mu_0**2-2*mu_0*X_mean)/(2*(lambda_0+N))
    b_N = b_0 + 0.5*(expr_1 + expr_2)

    gamma_likalihood = gamma.pdf(tau, a_N, scale=1/b_N)
    gauss_likelihood = norm.pdf(mu,mu_N,1/(lambda_N*tau)**0.5)
    posterior = gamma_likalihood*gauss_likelihood

    return posterior


np.random.seed(10)
# Draw samples from a Gaussian
mu, tau = 0, 0.1 # mean and precision of the gaussian
N = 100
X = np.random.normal(mu, 1/tau**0.5, N)
X_mean = np.sum(X)/N

# Initialize the parameters of the priors
a_0 = 0
b_0 = 0
lambda_0 = 0
mu_0 = 0
a_N, b_N, lambda_N, mu_N = iterations(a_0, b_0, lambda_0, mu_0, X, X_mean)


# Plot: On the x-axis mu, on the y-axis tau, on the z-axis q(mu,tau)

# Approximate distribution

mu_range = np.linspace(mu-1/(tau**0.5), mu+1/(tau**0.5), num=100)
tau_range = np.linspace(0, 4, num=100)

mu_range = np.linspace(-5,5, num=100)
tau_range = np.linspace(-5, 5, num=100)

R = len(mu_range)
C = len(tau_range)
Z = np.empty((R,C))

cont_x = cont_y = 0
for mu in mu_range:
    cont_y = 0
    print(cont_x)
    for tau in tau_range:
        Z[cont_y,cont_x] = norm.pdf(mu, mu_N, 1/lambda_N**0.5)*gamma.pdf(tau,a=a_N,loc=0,scale=1/b_N)
        cont_y += 1
    cont_x += 1
print('cont_x,cont_y',cont_x,cont_y)

xticks = np.round(mu_range,3)
yticks = np.round(tau_range,3)


plot_ = plt.contour(xticks, yticks, Z, colors='green')


# Exact posterior comparison

cont_x = cont_y = 0
for mu in mu_range:
    cont_y = 0
    print(cont_x)
    for tau in tau_range:
        Z[cont_y,cont_x] = exact_posterior( mu, tau, a_0, b_0, lambda_0, mu_0, X, X_mean)
        cont_y += 1
    cont_x += 1
print('cont_x,cont_y',cont_x,cont_y)

xticks = np.round(mu_range,3)
yticks = np.round(tau_range,3)


plot_ = plt.contour(xticks, yticks, Z, colors='blue')
plt.xlabel('mu')
plt.ylabel('tau')
plt.ylim([0,.3])
plt.xlim([-.5,1])
plt.show()
```

```
import numpy as np
from scipy.stats import multivariate_normal, poisson
import matplotlib.pyplot as plt


def EM(X, S, n_clusters, iter):

    # Initialize the mean, covariance, pi and the rates
    aux = []
    for i in range (n_clusters):
        aux.append(i+1)
    aux = np.array(aux)
    rates = (np.mean(S)*np.ones(n_clusters))/aux

    pi = np.ones(n_clusters)/n_clusters
    mu = np.random.randint(min(X[:,0]),max(X[:,0]),size=(n_clusters,len(X[0])))
    cov = np.zeros((n_clusters,len(X[0]),len(X[0])))
    for D in range(n_clusters):
        np.fill_diagonal(cov[D],3)


    for i in range(iter):

        # E_step from EM. Calculate responsabilities so as to update parameters in the M_step

        r = np.zeros((len(X),len(cov)))
        den = 0
        for k in range(n_clusters):
            den += pi[k]*multivariate_normal(mean=mu[k],cov=cov[k]).pdf(X)*poisson(rates[k]).pmf(S[k])


        for k in range(n_clusters):
            num = pi[k]*multivariate_normal(mean=mu[k],cov=cov[k]).pdf(X)*poisson(rates[k]).pmf(S[k])
            r[:,k] = num/den



        # M_step from EM. Calculate m_k. with m_k and the responsabilities, update the parameters pi,
mu, sigma

        pi = []
        mu = []
        cov = []
        rate = []
        for k in range(len(r[0])):
            m_k = np.sum(r[:,k],axis=0)
            mu_k = np.sum(X*r[:,k].reshape(len(X),1),axis=0)*(1/m_k)
            mu.append(mu_k)
            cov.append(((1/m_k)*np.dot((np.array(r[:,k]).reshape(len(X),1)*(X-mu_k)).T,(X-mu_k))))
            pi.append(m_k/np.sum(r))
            rate.append((1/m_k)*np.sum(S*r[:,k].reshape(len(S),1),axis=0))

    return mu, cov, pi, rates


# Reference:  https://python-
course.eu/expectation_maximization_and_gaussian_mixture_models.php#Unsupervised-learning:-Clustering:-
Gaussian-Mixture-Models-(GMM)
# np.random.seed(100)

from io import StringIO
f = open("X.txt", "r")
file = f.read()
c = StringIO(file)
data = np.loadtxt(c)
X = data

f = open("S.txt", "r")
file = f.read()
c = StringIO(file)
data = np.loadtxt(c)
S = data

n_clusters = 3
iter = 50
mu, cov, pi, rates = EM(X, S, n_clusters,iter)



# PLOT

# Create a grid for the plot
x,y = np.meshgrid(np.sort(X[:,0]),np.sort(X[:,1]))  # It can also be done with linespace
XY = np.array([x.flatten(),y.flatten()]).T
colors = ['green', 'blue', 'red', 'orange','purple']
plt.scatter(X[:,0],X[:,1])
for i in range(n_clusters):
    normal = multivariate_normal(mean=mu[i],cov=cov[i])

 plt.contour(np.sort(X[:,0]),np.sort(X[:,1]),normal.pdf(XY).reshape(len(X),len(X)),colors=colors[i],lin
ewidths=rates[i],alpha=0.3)


plt.show()

# Reference:  https://python-
course.eu/expectation_maximization_and_gaussian_mixture_models.php#Unsupervised-learning:-Clustering:-
Gaussian-Mixture-Models-(GMM)
```