DD2434 Machine Learning, Advanced Course Assignment 2

Baoqing She (baoqing@kth.se)

January 31, 2018

1 Graphical Models

I want to replace task2.2 by task2.6

2.1 Dependences in a Directed Graphical Model

Question 1: Which pairs of variables, not including X, are dependent conditioned on X?

If X is given, $A\&B$ are dependent. Since $C\&E$ and $D\&F$ are dependent no matter X is given or not, they will not be mentioned particularly.

Question 2: Which pairs of variables, not including X, are dependent, not conditioned on X?

If X is not given, A, B, C, D are obversely independent. The dependent pairs should be $A\&E$ $A\&F$ $B\&E$ $B\&F$ $E\&F$ $C\&E$ $D\&F$.

2.2 The Sum-HMM

Question 3: Implement the Sum-HMM, i.e., write your own code for it.

```
@author: SHE
"""
import numpy as np
import matplotlib.pyplot as plt
from collections import namedtuple

####
# Model definition
####

# Generates the probability of each outcome
# for each dice for all tables uniformly using
# the flat Dirichlet distribution [K*2*6]
def generateB(K):
    return np.random.dirichlet(np.ones(6), (K, 2))
    #uniform = np.ones((K, 2, 6)) / 6.
#return uniform
   #uniform[0]=np.random.dirichlet(np.ones(6), 2)
#return uniform


# Transition matrix
def generateA():
    return [[1/4., 3/4.], [3/4., 1/4.]]
```

```
# Generates the sequence for how a player
# trasitions to the next table [K]
def generateTableSeq(K):
    tables = [np.random.choice([0, 1])]
    for i in range(K-1):
        prob = 0.25
        if (tables[-1]):
            prob = 0.75
        tables.append(np.random.choice([0, 1], p=[prob, 1-prob]))

    return tables

# One player plays the game with an idividual
# trasition sequence ([K],1)]
Outcome = namedtuple('Outcome', ['X', 'S'])
def playOnce(B, K):
    tableSeq = generateTableSeq(K)
    x = []
    s = 0
    for k in range(K):
        obs = np.random.choice(int(6), p=B[k][tableSeq[k]]) + 1
        x.append(obs)

    return Outcome(x, s)

# Some observations in O.X are dropped '
# with probability p ([K],1)]
def hideObservations(O, p):
    return Outcome([None if np.random.uniform(0, 1) > p else x for x in O.X], O.S)

# Let (N) players play the game [N*(K,1)]
def play(N, B, K, p):
    return [hideObservations(playOnce(B, K), p) for n in range(N)]
```

Question 4: Provide data generated using at least three different sets of categorical dice distributions that provide reasonable tests for the correctness of your program .
Question 5: Motivate your test and why the result of it indicates correctness.

```
# Model Test (Question 4 and Question 5)
####
K = 10   # Num tables
N = 20   # Num players
p = 1.0  # 1-p is dropout probability
iterations = 100

B = generateB(K)

# Experimental probability
counts = np.zeros((K, 6))
for i in range(iterations):
    obs = play(N, B, K, p)
    for n in range(N):
        for k in range(K):
            if obs[n].X[k]:
                counts[k][obs[n].X[k]-1] += 1
```

```
exp_prob = counts / counts.sum(axis=1, keepdims=True)

# Theoretical probabiliy
theo_prob = np.array([(B[k][0]+B[k][1]) / 2. for k in range(K)])

# Plot
plt.hold(True)
plt.plot(exp_prob.flatten(),'ro-')
plt.plot(theo_prob.flatten(), 'bo-')
#plt.axis([0, 6, 0, 1])
plt.ylabel('Probability')
plt.show()
```

case 1: When setting dice have uniform distribution on all kinds of tables. As we can see from Fig.1, the blue dots are theoretical probability, which follows the uniform distribution with $1/6$ on all numbers. The red dots are obtained by using the date generated from models. The iteration is 8000 times, the theoretical and the practical probability are almost identical.

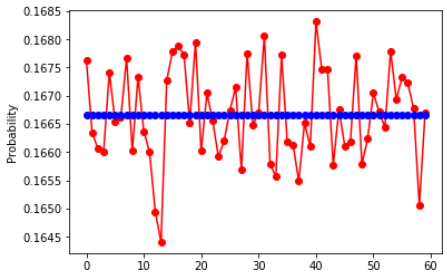

Figure 1: Distribution of output when dice are unbiased on every tables

case 2: When setting dice have uniform distribution only except the first table chosen. In Fig.2. The first six points have random probability and the rest have uniform distribution in both experimental and theoretical results.

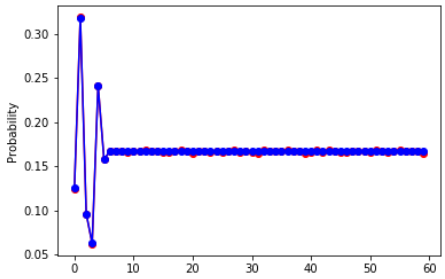

Figure 2: Distribution of output when only the first chosen table has biased dice

case 3: Set dice on all table have random distribution. The results are given in Fig.3

Assume the probability to choose the table is $\pi = (0.5, 0.5)$, and the transform matrix is given. Therefore, we get:

$$\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \begin{bmatrix} 0.25 & 0.75 \\ 0.75 & 0.25 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}.$$

Hence, the theoretical probability of the kth round of the experiment is:

$$P(X_k) = \frac{1}{2} P(X_k|t_k^1) + \frac{1}{2} P(X_k|t_k^2)$$

To verify the correctness of the model, the theoretical distribution of X_k is calculated by the formulation and preset distribution of dice above. Then the results are compared with the practical

distribution which is calculated by using the data from the model generated. After 100 iterations, the theoretical and practical probability are shown: As can be seen from the Fig,3, the red dots

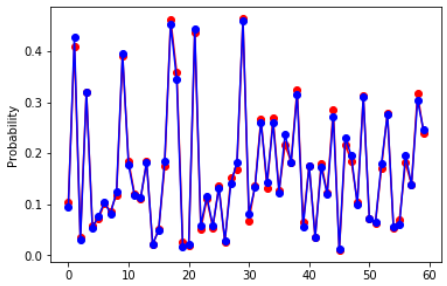

Figure 3: theoretical and practical output of model

represent the theoretical probability of output of this model and the line is the output we get after iterating the model we build 100 times. Two results show the same probability at the same number of the dice.

Question 6: Give polynomial time dynamic programming algorithm for computing $p(X_n^k = s; Z_k = t_k^i | s_n; x_n)$. Hint: a dice outcome is an integer between 1 and 6, so a sum s^n is an integer between K and 6K and, moreover, if a partial sum is associated with a state t_k^i, it is an integer between K and 6K.

The method is similar to how to train standard HMM. A_{ij} denotes the transition matrix. B_{ij}^k represents the state-output matrix. Therefore, we want to use forward-backward algorithm to get the probability we want.

1.Compute forward parameter [2] $\alpha_k(i,s) = p(sumX_1^n, ..., X_k^n = s, Z_k = t_k^i, X^n)$

1) For k = 1:

$$\alpha_1(i,s) = \begin{cases} \pi_i B^1(i, X_1^n), \ (if \ X_1^n \ is \ observed) \\ \pi_i B^1(i,s), \ (if \ X_1^n \ is \ not \ observed), \end{cases}$$

2) For k = 2,...,K:

$$\alpha_k(i,s) = \begin{cases} \sum_{i=1}^{2} \alpha_{k-1}(i, s - X_k^n) A_{(i,j)} B^k(j, X_k^n), \ (if \ X_k^n \ is \ observed) \\ \sum_{t=1}^{6} \sum_{i=1}^{2} \alpha_{k-1}(i, s-t) A_{(i,j)} B^k(j,t), (if \ X_k^n \ is \ not \ observed) \end{cases}$$

2. Compute backward algorithm [3] $\beta_k(i,s) = p(sum(X_{k+1}, ..., X_K^n) | Z_{k+1} = t_{t+1}^i, X^n)$

1) When k = 1:

$$\beta_K(i,s) = 1$$

2) When k = 1,..., K-1

$$\beta_k(i,s) = \begin{cases} \sum_{i=1}^{2} \beta_{k+11}(i, s - X_{k+1}^n) A_{(i,j)} B^{k+1}(j, X_{k+1}^n), \ (if \ X_k^n \ is \ observed) \\ \sum_{t=1}^{6} \sum_{i=1}^{2} \alpha_{k-1}(i, s-t) A_{(i,j)} B^{k+1}(j,t), \end{cases}$$

3. Compute forward-backward parameter[4] $p(X_k^n = s, Z_k = t_k^i | S^n, X^n)$.

The way to compute this value is multiply the backward parameters with forward parameters and then normalize the results. For more detail about it, refer to [2]. Since the forward and backward parameters are so small in this model, when implemented in code we use the normalized formate of these parameters.

```
## Forward algoritm
#def forward(K, A, B, X, S):
#    alpha = np.zeros(K, 2, 6*K)
#
#    # Alpha base case
#    if X[0]:
#        alpha[0][:,X[0]-1] = 1/2. * B[0][:,X[0]-1]
#    else:
#        alpha[0][:,:6] = 1/2. * B[0][:,:6]
#
#    # Alpha iteration
#    for k in range(1, K):
#        if X[0]:
#            for s in range(0, 6*K):
#                if s > X[k]:
#                    for l in range(2):
#                        for i in range(2):
#                            alpha[k][i][s] = alpha[k-1][i][s-X[K]-1] * A[l][i] * B[k][i][X[k]-1]
#        else:
#            for l in range(2):
#                for i in range(2):
#                    for s in range(0, 6*K):
#                        alpha[k][i][s] = alpha[k-1][i][s-] * A[l][i] * B[k][i][X[k]-1]
```

Question 7: Implement this DP algorithm, test it, in particular for varying values of p, and, finally, motivate your tests and why the result of it indicates correctness.

As the value p increases,the value of $p(X_k^n, Z_k = t_k^i | s^n, x^n)$ increase and get closer to the value when all the X are observable.

2.3 Simple VI

Question 8: Implement the VI algorithm for the variational distribution in Equation (10.24) in Bishop.

```
from __future__ import division
import numpy as np
from scipy.stats import norm, gamma
import matplotlib.pyplot as plt

#parameter set
mu_0 = 7
lambda_0 = 2
a_0 = 50
b_0 = 13
N = 1000
#generate data
X = np.random.normal(1,9,N)
mean = np.mean(X)

#compute parameter
mu_N = (lambda_0 * mu_0 + N * mean)/(lambda_0 +N)
a_N = a_0 + N/2
#initialize parameter
lambda_N = np.random.uniform(-10,10)
#updata parameter
```

```
sum_square = np.sum(X ** 2)
X_sum = sum(X)
def b_n(lambda_n):
    return b_0 + 0.5*((lambda_0 + N)*(1/lambda_n + mu_N **2)-2*(lambda_0 * mu_0 + X_sum) *mu_N +su

b_N = b_N_old = b_n(lambda_N)

epsilon = 0.000001
maxIter = 1000
for i in range(maxIter):
    b_N_old = b_N
    E_tau = a_N / b_N
    lambda_N = (lambda_0 + N) * E_tau
    b_N = b_n(lambda_N)
    print(lambda_N, b_N)
    if abs(b_N - b_N_old) < epsilon:
        break

x = np.linspace(0.8, 1.2, 1000)
Norm = norm.pdf(x, mu_N, 1/lambda_N)
y = np.linspace(0.01, 0.02, 1000)
Gamma = gamma.pdf(y, a_N, scale = 1/b_N)

Z = np.outer(Gamma, Norm)
Z_norm = Z / np.sum(Z)

plt.contourf(x, y, Z_norm, 20, alpha = 0.75, cmap = plt.cm.hot)
plt.xlabel( 'mu ' , fontsize = 15)
plt.ylabel( 'tau' , fontsize = 15)
plt.show()

print(a_N, b_N, mu_N, lambda_N)
```

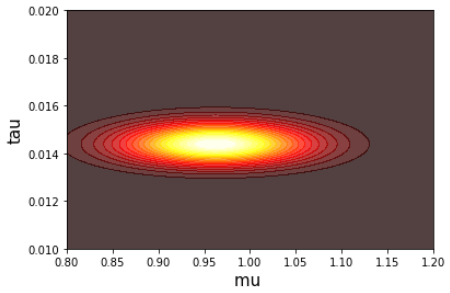

Figure 4: posterior estimated by VI algorithm

Question 9: Describe the exact posterior

Since $p(D|\tau, \mu)$ and $p(\mu|\tau)$ are Gaussian distribution and $p(\tau)$ is Gamma distribution, the exact

posterior is a Gaussian Gamma distribution:

$$
\begin{aligned}
p(\tau,\mu|D) &= \frac{p(D|\tau,\mu)p(\mu|\tau)p(\tau)}{p(D)} \\
&= P(D|\mu,\tau) + P(\mu|\tau) + P(\tau) + const \\
&= ln(\frac{\tau}{2\pi})^{N/2} exp[-\frac{\tau}{2}\sum_{n=1}^{N}(x_n-\mu)^2] + ln(\frac{\lambda_0\tau}{2\pi})^{\frac{1}{2}} exp[-\frac{\lambda_0\tau}{2}(\mu-\mu_0)^2] \\
&+ lnGam(\tau|a_0,b_0) + const \\
&= -\frac{\lambda^*\tau}{2}(\mu-\mu^*)^2 + \frac{\tau}{2}\frac{(N\hat{x}+\lambda_0)^2}{N+\lambda_0} - (0.5N\hat{x}^2 + \frac{\lambda_0}{2}\mu_0 + b_0)\tau \\
&= -\frac{\lambda^*\tau}{2}(\mu-\mu^*)^2 - \tau[0.5\sum_{n=1}^{N}(x_n-\hat{x})^2 + 0.5\frac{N\lambda_0(\hat{x}-\mu_0)^2}{N+\lambda_0} + b_0] + (a^*-1)ln(\tau) + const \\
&= -\frac{\lambda^*\tau}{2}(\mu-\mu^*)^2 - b^*\tau + (a^*-1)ln(\tau) + cost
\end{aligned}
$$

where, the exact value for the parameters of Gaussian and gamma distribution are given:

$$\mu^* = \frac{\sum_{n=1}^{N} x_n + \lambda_0\mu_0}{\lambda_0 + N}$$

$$\lambda^* = N + \lambda_0$$

$$a^* = \frac{N+1}{2} + a_0$$

$$b^* = \frac{1}{2}(\sum_{n=1}^{N}(x_n-\bar{x})^2 + \frac{N\lambda_0(\bar{x}-\mu_0)^2}{N+\lambda_0}) + b_0$$

Question 10: Compare the variational distribution with the exact posterior. Run the inference for a couple of interesting cases and describe the difference.

Implement the exact posterior using the results from question 9. The values of $\{\mu, \lambda, a, b\}$ from VI algorithm. The comparison of the exact parameters and estimated parameters are shown in the table below:

parameter	μ	λ	a	b
exact value	0.13832	100	60.5	145.46
VI value	0.13832	70	60	146.68

Code:

```
# compute exact posterior
mu_e = (X_sum + lambda_0 * mu_0)/(lambda_0 + N)
lambda_e = N + lambda_0
a_e = (N+1)/2. +a_0
X_sub = [i-mean for i in X]
X_squre = [j**2 for j in X_sub]
b_e = b_0 + 0.5*(sum(X_squre) + (N*lambda_0*((mean - mu_0) ** 2))/(N + lambda_0))

print('e',  mu_e, lambda_e, a_e, b_e)
print('a', mu_N, lambda_N, a_N, b_N)

x_e = np.linspace(0.8, 1.2, 1000)
Norm_e = norm.pdf(x_e, mu_e, 1/lambda_e)
y_e = np.linspace(0.01, 0.02, 1000)
Gamma_e = gamma.pdf(y_e, a_e, scale = 1/b_e)

Z_e= np.outer(Gamma_e, Norm_e)
```

```
Z_norme = Z_e / np.sum(Z_e)

plt.contourf(x_e, y_e, Z_norme, 20, alpha = 0.75, cmap = plt.cm.hot)
plt.xlabel( 'mu ' , fontsize = 15)
plt.ylabel( 'tau' , fontsize = 15)
plt.show()
```

2.6 Variational Inference

Question 15: Present the algorithm written down in a formal manner (using both text and mathematical notation, but not pseudo code).

Since $S_{rc} = X_r + Y_c$, S_{rc} follow Gaussian distribution when mean is $\mu_r + \xi_r^{-1} + \xi_c^{-1}$.Then, the likelihood can be expressed as:

$$p(S|\mu_1, ..., \mu_R, \xi_1, ..., \xi_C) = \prod_{c=1}^{C} \prod_{r=1}^{R} \frac{1}{\sqrt{2\pi(\lambda_r^{-1} + \lambda_c^{-1})}} e^{-\frac{(S_{rc} - \mu_r - \xi_c)^2}{2(\lambda_r^{-1} + \lambda_c^{-1})}}$$

where,

$$\begin{cases} p(\mu_r) = \sqrt{\frac{\lambda}{2}} e^{-\frac{\lambda}{2}(\mu_r - \mu)^2}, \\ p(\xi_c) = \sqrt{\frac{\tau}{2}} e^{-\frac{\tau}{2}(\xi_c - \xi)^2}, \end{cases}$$

According to [1], the optimal solution $q_j^*(Z_j)$ should be:

$$q_j^*(Z_j) = E_{i \neq j}[lnp(X, Z)] + const$$

Here, $Z = \{\mu_1, ..., \mu_R, \xi_1, ..., \xi_C\}$, $X = S_{rc}$. Therefore, the optimal solution for $q(\mu_r)$:

$$\begin{aligned} lnq^*(\mu_r) &= E_{Z \neq \mu_r}[ln(p(S|Z)p(Z))] + const \\ &= E_{Z \neq \mu_r}[-\sum_{c=1}^{C} \frac{1}{2(\lambda_r^{-1} + \lambda_c^{-1})}(S_{rc} - \mu_r - \xi_c)^2 - \frac{\lambda}{2}(\mu_r - \mu)^2] + const \\ &= E_{Z \neq \mu_r}[-\sum_{c=1}^{C} \frac{1}{2(\lambda_r^{-1} + \lambda_c^{-1})}(\mu_r^2 + 2\mu_r\xi_c - 2S_{rc}\mu_r)^2 - \frac{\lambda}{2}(\mu_r^2 - 2\mu\mu_r)] + const \\ &= [-\frac{\lambda}{2} - \sum_{c=1}^{C} \frac{1}{2(\lambda_r^{-1} + \lambda_c^{-1})}]\mu_r^2 + [\lambda\mu + \sum_{c=1}^{C} \frac{S_{rc} - E[\xi_c]}{2(\lambda_r^{-1} + \lambda_c^{-1})}]\mu_r + const \end{aligned}$$

Hence, $lnq^*(\mu_r)$ is a Gaussian distribution, the mean and variance are given:

$$\begin{aligned} \mu_{\mu_r}^N &= [\lambda\mu + \sum_{c=1}^{C} \frac{S_{rc} - E[\xi_c]}{2(\lambda_r^{-1} + \lambda_c^{-1})}]/[\lambda - \sum_{c=1}^{C} \frac{1}{\lambda_r^{-1} + \lambda_c^{-1}}] \\ \lambda_{\mu_r}^N &= \lambda - \sum_{c=1}^{C} \frac{1}{\lambda_r^{-1} + \lambda_c^{-1}} \end{aligned}$$

Using the similar method, we get the distribution of ξ_c are also a Gaussian distribution:

$$lnq^*(\xi_c) = [-\frac{\tau}{2} - \sum_{r=1}^{R} \frac{1}{2(\lambda_r^{-1} + \lambda_c^{-1})}]\xi_c^2 + [\tau\mu + \sum_{r=1}^{R} \frac{S_{rc} - E[\mu_r]}{2(\lambda_r^{-1} + \lambda_c^{-1})}]\xi_c + const$$

Therefore, the mean and variance are given:

$$\begin{aligned} \mu_{\xi_c}^N &= [\tau\mu + \sum_{r=1}^{R} \frac{S_{rc} - E[\mu_r]}{\lambda_r^{-1} + \lambda_c^{-1}}]\ [\tau + \sum_{r=1}^{R} \frac{1}{\lambda_r^{-1} + \lambda_c^{-1}}] \\ \lambda_{\xi_c}^N &= \tau + \sum_{r=1}^{R} \frac{1}{\lambda_r^{-1} + \lambda_c^{-1}} \end{aligned}$$

From the results above, the variance of the distribution are all constant. The mean of row distribution and the mean of column distribution are dependent on each other. Therefore, we can use iteration to update the parameters.
Firstly, calculate both $\lambda_{\mu_r}^N$ and λ_{ξ_c} Then, initialize μ_r^N and ξ_c^N. Finally, use the formulation of $\mu_{\xi_c}^N$ and $\mu_{\mu_r}^N$ to update the value until convergence.

2.7 Variational Inference

Question 16: Present the algorithm written down in a formal manner (using both text and mathematical notation, but not pseudo code).

There are four kind of latent variable in this model, M ={μ^1, μ^2, ξ, Z}, where Z denote the sequence for table choosing. Since $S_k^n = X_k^n + Y_k^n$, $S_k^n = N(\mu_k^i + \xi_n, \lambda_k^{-1} + t^{-1})$. Therefore, the likelihood can be expressed as:

$$p(S|\mu^1, \mu^2, \xi, Z) = \prod_{n=1}^{N}\prod_{k=1}^{K} N(S_k^n|\mu_k^{Z_k} + \xi_n, \lambda_k^{-1} + t^{-1})$$

The prior of parameters are given:

$$p(\mu^1) = \prod_{k=1}^{k} N(\mu_k^1|\mu, \lambda^{-1})$$

$$p(\mu^2) = \prod_{k=1}^{k} N(\mu_k^2|\mu, \lambda^{-1})$$

$$p(\xi) = \prod_{n=1}^{N} N(\xi_n|\xi, t^{-1})$$

$$p(Z) = \frac{1}{2}(\frac{1}{4})^(K-1)\prod_{k=1}^{K-1} 3^I$$

where, $I = 1$ *when* $Z_k \neq Z_{k+1}$, $I = 0$ *when* $Z_k = Z_{k+1}$

Therefore, the optimal solution for $q(\xi_n)$ is:

$$\begin{aligned} lnq^*(\xi_n) &= lnE_{M\neq\xi_n}[p(S|P)p(P)] + const \\ &= E_{M\neq\xi_n}[\sum_{n=1}^{N}\sum_{k=1}^{K}] - \frac{(S_k^n - \mu_k^{Z_k} - \xi_n)^2}{2(\lambda_k^{-1} + t^{-1})} - \frac{t}{2}(\xi_n - \xi)^2 \\ &= [-\frac{t}{2} - \sum_{n=1}^{N}\sum_{k=1}^{K}\frac{1}{2(\lambda_k^{-1} + t^{-1})}]\xi_n^2 + [t\xi + \sum_{n=1}^{N}\sum_{k=1}^{K}\frac{S_k^n - E[u_k^{Z_k}]}{\lambda_k^{-1} + t^{-1}}]\xi_n \end{aligned}$$

Hence, the $q(\xi_n)$ is a Gaussian distribution. The mean and variance are:

$$\begin{aligned} \mu_{\xi_n}^N &= [t\xi + \sum_{n=1}^{N}\sum_{k=1}^{K}\frac{S_k^n - E[u_k^{Z_k}]}{\lambda_k^{-1} + t^{-1}}]/[t + \sum_{n=1}^{N}\sum_{k=1}^{K}\frac{1}{\lambda_k^{-1} + t^{-1}}] \\ \lambda_{\xi_n}^N &= t + \sum_{n=1}^{N}\sum_{k=1}^{K}\frac{1}{\lambda_k^{-1} + t^{-1}} \end{aligned}$$

Using similar method, the distribution of $q(\mu_k^1)$ and $q(\mu_k^2)$ can be obtained. They both follow Gaussian distribution.

$$lnq(\mu_k^1) = [-\frac{\lambda}{2} - \sum_{n=1}^{N}\sum_{Z_k=1}\frac{1}{2(\lambda_k^{-1} + t^{-1})}](\mu_k^1)^2 + [\lambda\mu + \sum_{n=1}^{N}\sum_{Z_k=1}\frac{S_k^n - E[\xi_n]}{\lambda_k^{-1} + t^{-1}}]\mu_k^1$$

Therefore, the mean and the variance are:

$$\mu_{\mu_k^1}^N = [\lambda\mu + \sum_{n=1}^{N}\sum_{Z_k=1}\frac{S_k^n - E[\xi_n]}{\lambda_k^{-1}+t^{-1}}]/[\lambda + \sum_{n=1}^{N}\sum_{Z_k=1}\frac{1}{\lambda_k^{-1}+t^{-1}}]$$

$$\lambda_{\mu_k^1} = \lambda + \sum_{n=1}^{N}\sum_{Z_k=1}\frac{1}{2(\lambda_k^{-1}+t^{-1})}$$

In the similar way:

$$\mu_{\mu_k^2}^N = [\lambda\mu + \sum_{n=1}^{N}\sum_{Z_k=1}\frac{S_k^n - E[\xi_n]}{\lambda_k^{-1}+t^{-1}}]/[\lambda + \sum_{n=1}^{N}\sum_{Z_k=1}\frac{1}{\lambda_k^{-1}+t^{-1}}]$$

$$\lambda_{\mu_k^2} = \lambda + \sum_{n=1}^{N}\sum_{Z_k=2}\frac{1}{2(\lambda_k^{-1}+t^{-1})}$$

Using the formulation above, we can iteratively compute $q(\xi_n), q(\mu_k^1), q(\mu_k^2)$.

References

[1] Christopher M. Bishop. Pattern Recognition and Machine Learning

[2] https://www.youtube.com/watch?v=MPmrFu4jFk4&t=139s

[3] https://www.youtube.com/watch?v=jwYuki9GgJo

[4] https://www.youtube.com/watch?v=7zDARfKVm7s