Assignment 2

Marcus Lindström
880619-0453

January 6, 2019

2.1 Knowing the rules

Question 2.1.1:

Yes I have.

Question 2.1.2:

I've not been collaborating with anyone regarding the problem formulations.

Question 2.1.3:

No, I have not.

2.2 Dependencies in a Directed Graphical Model

For this part I chose to always assume that the colored nodes in the assignment pictures were unobserved i.e. $X_{r,c}^n$ in 2.2.4-2.2.6 and the set X in 2.2.7-2.2.9, if the task does not explicitly state otherwise.

Question 2.2.4:

Yes.

Question 2.2.5:

No.

Question 2.2.6:

$A = \{\mu_{r,c}, \mu_{r+1,c}, \mu_{r-1,c}, \mu_{r,c+1}, \mu_{r,c-1}\}$.

Question 2.2.7:

No.

Question 2.2.8:

No.

Question 2.2.9:

$B = \{C^n : n \in [N]\} \cup \{Z_m^n : n \in [N], m \in [M]\}$.

2.3 Likelihood of a tree GM

Question 2.3.10:

To calculate $P(\beta|T, \Theta)$, we first denote:

$$s(u, x_i) = P(x_{\downarrow u \cap L} | X_u = x_i), \tag{1}$$

i.e. the probability of a subtree in T rooted at u given that u has the value x_i. This is evaluated for all u and x_i recursively in a postorder fashion throughout the tree as follows:

$$s(u, x_l) = \begin{cases} 1 \text{ if } x_u = x_l \\ 0 \text{ otherwise} \end{cases}, \quad \forall u \in L, \tag{2}$$

$$s(u, x_i) = \prod_v \sum_j p(X_v = x_j | u = x_i) s(v, j), \quad \forall u \in V. \tag{3}$$

Where $\text{Pa}(v) = u$ and x_l denotes the observed value of leaf u. These are stored in each tree node object as arrays with length i (utilizing the concept of dynamical programming), and can then be used in order to compute

$$P(\beta|T, \Theta) = \sum_i s(r, x_i) P(X_r = x_i). \tag{4}$$

Where r denotes the root.

Question 2.3.11:

The likelihood computation was done for all trees given in the supplied data for all their respective samples resulting in the 27×3 values listed in Table 1. The code used to generate the likelihood of one tree was an adaption of the supplied code where a minor change was done to the Node class and also the inclusion of two more functions, these are supplied in appendix A. The function 'treeLikelihood' takes as a single input the tree root and generates the likelihood.

Table 1: Likelihood of the trees for different leaf set samples supplied in the assignment data.

Tree Name \ Sample Number	**1**	**2**	**3**
tree_k_3_md_3_mb_7_alpha_[0.1 0.5 0.5]	0.0155753676376	0.088023946612	0.132566342716
tree_k_2_md_5_mb_5_alpha_[0.5 0.5]	9.54808310286e-09	1.70196002223e-08	1.98877723494e-08
tree_k_3_md_10_mb_2_alpha_[0.1 0.1 0.5]	0.626315421676	0.0395215515401	0.626315421676
tree_k_3_md_3_mb_5_alpha_[0.1 0.1 0.5]	0.980132424063	0.980132424063	0.980132424063
tree_k_2_md_10_mb_2_alpha_[0.1 0.5]	0.887713912728	0.887713912728	0.887713912728
tree_k_5_md_3_mb_5_alpha_[1. 0.1 0.1 0.1 0.1]	0.969178646303	0.969178646303	0.969178646303
tree_k_2_md_5_mb_7_alpha_[0.1 0.1]	1.68177519885e-07	1.08421338523e-07	1.99629915369e-06
tree_k_5_md_10_mb_3_alpha_[0.1 0.1 1. 0.1 0.1]	0.470978623571	0.470978623571	0.512871879451
tree_k_2_md_3_mb_5_alpha_[0.1 0.1]	0.00313404991624	0.412272060248	0.412272060248
tree_k_3_md_5_mb_5_alpha_[0.1 0.5 0.1]	0.0970490659887	0.0154784896316	0.551756975511
tree_k_5_md_5_mb_3_alpha_[0.1 0.1 0.1 0.1 0.1]	0.0132414232261	0.000204224941913	0.00163860422448
tree_k_2_md_3_mb_3_alpha_[0.1 0.1]	0.381008332013	0.381008332013	0.615008301187
tree_k_2_md_10_mb_4_alpha_[0.1 0.1]	8.3402908101e-29	3.16982163034e-30	1.99587753423e-26
tree_k_2_md_5_mb_3_alpha_[0.1 0.1]	0.00118320383767	0.0127382652232	0.0233906447948
tree_k_3_md_5_mb_3_alpha_[1. 1. 0.1]	0.00942654885149	0.00116047367951	0.0034806316623
tree_k_5_md_5_mb_7_alpha_[0.1 0.1 0.1 0.5 0.1]	3.55174184376e-33	3.51890290718e-29	1.11323090791e-35
tree_k_2_md_3_mb_7_alpha_[0.1 0.5]	0.00138132794419	0.0244512198493	0.00229379539903
tree_k_3_md_5_mb_7_alpha_[0.5 0.1 0.1]	4.08500834532e-43	8.96183860932e-46	5.44033353243e-50
tree_k_3_md_10_mb_4_alpha_[0.1 0.1 0.1]	1.06182512223e-127	1.04982863414e-114	3.92169187603e-124
tree_k_5_md_3_mb_3_alpha_[0.1 0.5 0.1 0.1 0.1]	0.227582214428	0.227582214428	0.174149077642
tree_k_3_md_3_mb_3_alpha_[0.1 1. 1.]	0.27719738724	0.27719738724	0.111538941549
tree_k_5_md_10_mb_2_alpha_[1. 0.5 0.1 0.5 0.5]	0.0405915901623	0.0372884579238	0.153115682736
tree_k_2_md_10_mb_3_alpha_[1. 0.1]	0.150394889842	0.0177822750097	0.163041712878
tree_k_3_md_10_mb_3_alpha_[0.5 0.1 0.5]	0.0637245109469	0.538856832905	0.309363515032
tree_k_5_md_3_mb_7_alpha_[0.1 0.1 0.5 1. 0.1]	7.29290846901e-05	8.22304199798e-05	4.5918149222e-07
tree_k_5_md_10_mb_4_alpha_[0.1 0.1 0.1 0.1 0.1]	2.44568054619e-299	6.06300242703e-289	8.74557857658e-304
tree_k_5_md_5_mb_5_alpha_[0.5 0.5 0.1 1. 1.]	4.4826857152e-38	2.0737157063e-33	1.95182672736e-32

3 Simple VI only for E level

Question 2.4.12:

The VI-algorithm to approximate the posterior was implemented in python with code supplied in appendix B.

Question 2.4.13:

The exact posterior is given by a Gaussian-Gamma distribution $\mathcal{NG}(\mu, \tau|\mu_n, \lambda_n, a_n, b_n)$ where

$$\mu_n = \frac{\lambda_0\mu_0 + n\bar{x}}{\lambda_0 + n}, \tag{5}$$

$$\lambda_n = \lambda_0 + n, \tag{6}$$

$$a_n = a_0 + n/2, \tag{7}$$

$$b_n = b_0 + \frac{1}{2}\left(\sum_{i=1}^{n}(x_i - \bar{x})^2 + \frac{\lambda_0 n(\bar{x} - \mu_0)^2}{(\lambda_0 + n)}\right). \tag{8}$$

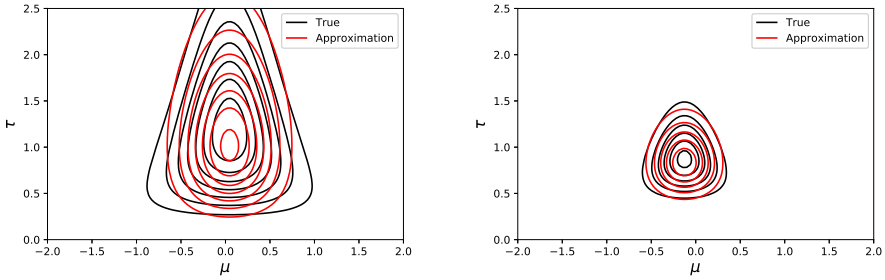
It is derived from realizing that it is proportional to the likelihood times the prior and that the used prior is conjugate, then identifying all parameters after some rearrangement. A full derivation can be found here for example.

Question 2.4.14:

Below follows three cases where approximate posterior distributions $p(\mu, \tau|\mathbf{X})$ were inferred using VI until convergence. In each case, different parameter setups were used to tweak both the τ-priors and also the data generating normal distribution.

Case 1:

- Prior: $a_0 = 1$, $b_0 = 0.001$, $\lambda_0 = 1$, $\mu_0 = 0$
- Data: $\mu = 0$, $\tau = 1$

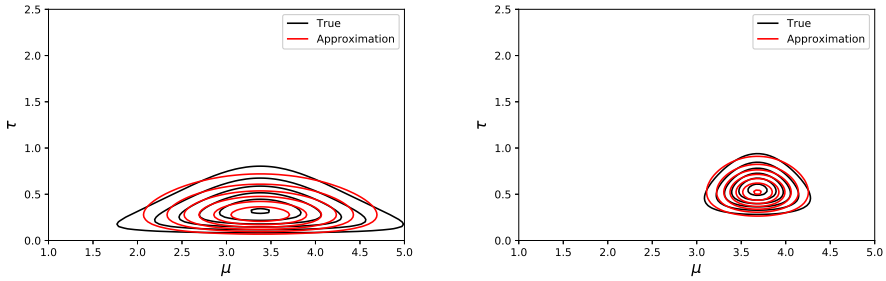


(a) $n = 5$ (b) $n = 20$

Figure 1: Comparison of the true and approximate posterior for case 1.

Case 2:

- Prior: $a_0 = 1$, $b_0 = 0.001$, $\lambda_0 = 1$, $\mu_0 = 0$
- Data: $\mu = 4$, $\tau = 1$

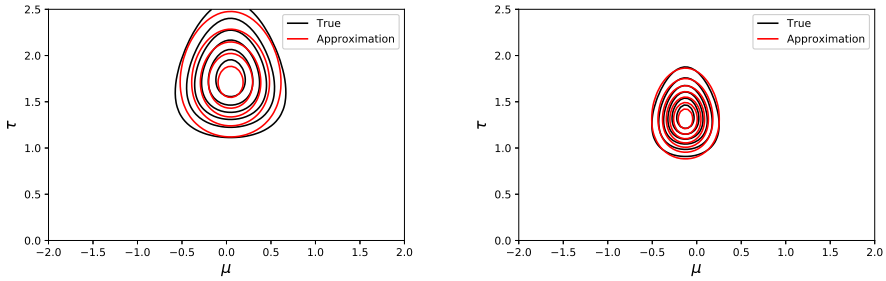


(a) $n = 5$ (b) $n = 20$

Figure 2: Comparison of the true and approximate posterior for case 2.

Case 3:

- Prior: $a_0 = 20$, $b_0 = 10$, $\lambda_0 = 1$, $\mu_0 = 0$
- Data: $\mu = 0$, $\tau = 1$



(a) $n = 5$ (b) $n = 20$

Figure 3: Comparison of the true and approximate posterior for case 3.

We see that in both case 1 and 2 the true posterior and the approximate (somewhat) catches the parameters accurately whereas case 1 is closest, (note also that the same random seed was used in all cases). This is due to how we encode our prior belief in the parameters, case 1 had a better "guess" of the data mean encoded in its prior, which however mostly affect the learned precision value. This is partly a consequence of the distribution being constrained to a finite probability mass and has to become more compact in one direction if it's to stretch in another.

If we look at figure 4, we see how the precision Gamma-prior was modeled in case 1 & 2 compared to case 3. The formers were modeled with the least informative prior, and adding more data was indeed informative, which we can see as the distribution becoming more compact. Case 3 used a considerably

more informative prior, placing much of its probability mass on a wrong value. Adding fifteen more data samples did not do as much in terms of providing more information compared to earlier cases, which suggests slower (at least initial) learning rates.

The approximate posterior mean always coincides with that of the true one, which does not come as a surprise since they are given by the same expression.

It's also apparent that the true posterior does no factorize in a way used in the VI-algorithm, this is easiest to see in case 1 & 2 where they differ quite a lot. The approximate is also more compact in each case, which is a general result of factorized variational approximations of posterior distributions according to Bishop p. 467.

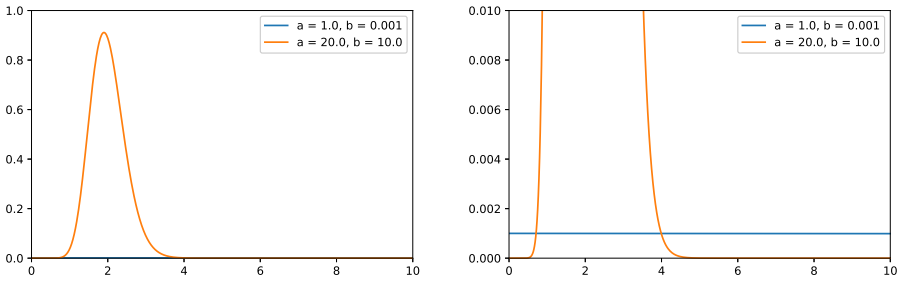


Figure 4: Comparison of the two types of Gamma-priors used, the right picture is a zoomed in version of the left.

A Code for 2.3

```
class Node:

    def __init__(self, name, cat):


        self.cat         = []
        for c in cat:
            self.cat.append(c)
        self.ancestor    = None
        self.descendants = []
        self.sample      = None
        self.smaller     = np.zeros(len(cat[0]))
        self.name        = name


def makeSmaller(root):

''' Generates s(u,x_i) for each node
and stores them in self.smaller '''
    if root:
        for child in root.descendants:
            makeSmaller(child)
        if root.descendants == []:
            root.smaller[int(root.sample)] = 1
        else:
            for i in range(len(root.cat[0])):
                temp1 = 1.0
                for child in root.descendants:
                    temp2 = 0.0
                    for j in range(len(child.cat[0])):
                        temp2 += child.cat[i][j]*child.smaller[j]
                    temp1 *= temp2
                root.smaller[i] = temp1

def treeLikelihood(root):

''' Calls makeSmaller to prepare
tree through evaluating each s(u,x_i),
then returns the tree likelihood '''
    makeSmaller(root)
    likelihood = 0.0
    for i in range(len(root.cat[0])):
        likelihood += root.smaller[i]*root.cat[0][i]
    return likelihood
```

B Code for 2.4

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.special import gamma as g

# Regular NGa - pdf, used to describe the exact posterior for mu and tau
def normalGamma(m, l, a, b):
    C = (b**a)*np.sqrt(l) / (g(a) * np.sqrt(2*np.pi))
    return lambda x, tau:C*tau**(a-0.5)*np.exp(-b*tau-0.5*l*tau*(x-m)**2)

# Precision prior
def gamma(a,b):
    C = b**a/g(a)
    return lambda tau: C * tau**(a-1)*np.exp(-b*tau)

# Mean prior
def normal(mu,sigma):
    C = np.sqrt(2*np.pi*sigma**2)
    return lambda m: C * np.exp(-(m-mu)**2*0.5/sigma**2)

# VI approximated posterior
def vi(a,b,mu,l):
    return lambda x, tau: normal(mu,1./np.sqrt(l))(x)*gamma(a,b)(tau)

np.random.seed(1)
N = 5

# Generate Gaussian univariate samples
X  = np.random.normal(0, 1, N)
m  = np.mean(X)
v  = np.var(X)

# Initialize prior
a_0 = 1; b_0 = 1; l_0 = 1; mu_0 = 0

# Calculate true posterior
a_t  = a_0 + N/2
l_t  = l_0 + N
mu_t = (l_0*mu_0 + N*m) / (l_0 + N)
b_t  = b_0 + 1./2*(N*v + (l_0*N*(m - mu_0)**2)/(l_0 + N))
x = np.linspace(-2, 2, 100)
y = np.linspace(0, 5, 100)
Z1 = normalGamma(mu_t, l_t, a_t, b_t)(*np.meshgrid(x, y))

# VI - parameters
mu_v = (l_0*mu_0 + N*m)/(l_0 + N)
a_v  = a_0 + (N+1)/2.
```

```
# Initial guess
l_v = 1

# 100 iterations which strongly suggested convergence after some testing
for _ in range(100):
        b_v  = b_0 + 0.5 * ((l_0 + N) * (1./l_v + mu_v**2)
                            - 2 * (l_0 * mu_0 + N*m) * mu_v
                            + np.sum(X**2) + l_0*mu_0**2)
        l_v  = (l_0 + N)*(a_v/b_v)

# Plotting
fig, ax = plt.subplots()
Z2 = vi(a_v, b_v, mu_v, l_v)(*np.meshgrid(x, y))
cntr1 = ax.contour(x, y, Z1, colors='k')
cntr2 = ax.contour(x, y, Z2, colors='r')
h1,_ = cntr1.legend_elements()
h2,_ = cntr2.legend_elements()
ax.legend([h1[0], h2[0]], ['True', 'Approximation'])
ax.set_xlabel('$\mu$', fontsize=15)
ax.set_ylabel('$\\tau$', fontsize=15)
plt.savefig('Case3a.eps', bbox_inches='tight')
plt.show()
```