KTH Royal Institute of Technology

Machine Learning Advanced Course

Assignment 2

Castellano Giovanni

October 3, 2022

2.1 Dependencies in a Directed Graphical Model

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

2.2 Likelihood of a Tree Graphical Model

The algorithm is based on the algorithm studied in module 7 lecture 6 where we studied how to compute the probability that a node X_u has a certain value after observing the values of the leaves. In particular, during the lecture we defined two probabilities; one concerning the nodes above X_u, and the second one concerning the nodes below X_u. To compute the probability of the leaves I computed the value of the second function on the root node for all possible values that the root can assume. Subsequently, I multiplied the results for the probability that the root assumes those values and summed them up.

- Small Tree:
  0.009786075668602368
  0.015371111945909397
  0.02429470256988136
  0.005921848333806081
  0.016186321212555956

- Medium Tree:
  1.7611947348905713e-18
  2.996933026124685e-18
  2.891411201505415e-18
  4.6788419411270006e-18
  5.664006737201378e-18
- Large Tree:
  3.63097513075208e-69
  3.9405421986921234e-67
  5.549061147187144e-67
  9.89990102807915e-67
  3.11420969368965e-72

2.3 Simple Variational Inference

Exact posterior

Suppose to have some data $\mathbf{X} = x_1, ..., x_n \sim Normal(\mu, \lambda^{-1})$, where $\lambda = 1/\sigma^2$. Suppose moreover, that we do not know the exact values of the parameters μ and λ but we know that $(\mu, \lambda) \sim NormalGamma(m, c, a, b)$, and therefore:

$$\mu|\lambda \sim Normal(m, (c\lambda)^{-1}) \tag{1}$$

$$\lambda \sim Gamma(a, b) \tag{2}$$

$$P(\mu, \lambda) = P(\mu|\lambda)P(\lambda) = Normal(\mu|m, (c\lambda)^{-1}) * Gamma(\lambda|a, b) \tag{3}$$

What we want to prove here, is that the NormalGamma is a conjugate prior for the Normal distribution, i.e.:

$$P(\mu, \lambda|\mathbf{X}) \sim NormalGamma(M, C, A, B) \tag{4}$$

We have:

$$NormalGamma(\mu, \lambda|m, c, a, b) = Normal(\mu|m, (c\lambda)^{-1})Gamma(\lambda|a, b) \tag{5}$$

$$NormalGamma(\mu, \lambda|m, c, a, b) = \sqrt{\frac{c\lambda}{2\pi}} e^{-0.5c\lambda(\mu-m)^2} \frac{b^a}{\Gamma(a)} \lambda^{a-1} e^{-b\lambda} \tag{6}$$

$$\sqrt{\frac{c\lambda}{2\pi}} e^{-0.5c\lambda(\mu-m)^2} \frac{b^a}{\Gamma(a)} \lambda^{a-1} e^{-b\lambda} \propto \lambda^{a-1/2} e^{-0.5\lambda(c\mu^2 - 2cm\mu + cm^2 + 2b)} \tag{7}$$

Where we developed the square and joined the exponentials; moreover for any x:

$$Normal(x|\mu, \lambda^{-1}) = \sqrt{\frac{\lambda}{2\pi}} e^{-0.5\lambda(x-u)^2} \tag{8}$$

$$\sqrt{\frac{\lambda}{2\pi}} e^{-0.5\lambda(x-u)^2} \propto \lambda^{1/2} e^{-0.5\lambda(\mu^2 - 2x\mu + x^2)} \tag{9}$$

thus:

$$P(\mu, \lambda|\mathbf{X}) \propto P(\mu, \lambda) * P(\mathbf{X}|\mu, \lambda) \tag{10}$$

using equation 7 and equation 9 we get:

$$P(\mu, \lambda)P(\mathbf{X}|\mu, \lambda) \propto \lambda^{a-1/2} e^{-0.5\lambda(c\mu^2 - 2cm\mu + cm^2 + 2b)} \lambda^{n/2} e^{-0.5\lambda(n\mu^2 - 2(\sum_i^n x_i)\mu + \sum_i^n x_i^2)} \tag{11}$$

where we are supposing that the data points are drawn independently.

$$P(\mu, \lambda)P(\mathbf{X}|\mu, \lambda) \propto \lambda^{a+n/2-1/2} e^{-0.5\lambda((c+n)\mu^2 - 2(cm + \sum_i^n x_i)\mu + cm^2 + 2b + \sum_i^n x_i^2)} \tag{12}$$

therefore:

$$P(\mu, \lambda)P(\mathbf{X}|\mu, \lambda) \propto \lambda^{A-1/2} e^{-0.5\lambda(C\mu^2 - 2(CM)\mu + CM^2 + 2B} \tag{13}$$

that is up to a constant a NormalGamma distribution, and we have defined:

$$A = a + n/2 \tag{14}$$

$$C = c + n \tag{15}$$

$$CM = cm + \sum_i^n x_i \tag{16}$$

$$CM^2 + 2B = cm^2 + 2b + \sum_i^n x_i^2 \tag{17}$$

Notice that equations 15, 16 and 17 contains 3 unknown values, and since they are exactly 3 equations, it is possible to join them and show that they hold for these values of M and B:

$$M = \frac{cm + \sum_i^n x_i}{c + n} \tag{18}$$

$$B = b + 0.5(cm^2 - CM^2 + \sum_i^n x_i^2) \tag{19}$$

substituting A, B, C and M in equation 13 we would get again 12. This proof is based on the proof provided by Rebecca C. Steorts and Lei Qian [1].

Approximation of the posterior with the VI algorithm

The following plots illustrate the prior probability of the parameters (green), the exact posterior (blue), and the approximated posterior (red). Moreover, we can observe in blue the parameters of the gaussian from which the points are drawn, and in orange the same parameters estimated from the generated data points.

To begin, figure 1 shows how the distributions appear when the posterior is approximated only using 2 data points. Moreover, in this case the prior is centered on the true parameters. In this case, the algorithm converged after 3 iterations. We notice that the center of the posterior is close to the center of the prior, and far from the estimated parameters (orange point). When we increase the number of data points (figure 2) from 2 to 50 however, we observe that the center of the posterior moves closer to the orange point; and If we increase the number of points from 50 to 500 (figure 3), we can notice that now the posterior is perfectly centered on the orange point, moreover, the two points are very close (this is due to the law of large numbers). It is also interesting that the variance of the posterior decreased as the number of points increased. In all the three cases, we observed a good approximation of the posterior. However, the approximation seems to get better as the number of data points increase.

In figure 4 we show an example where the approximation of the posterior is not very good. As we can observe, with this configuration of parameters, the prior distribution assumes a skewed shape; in particular, we notice some tails on the bottom. Such tails also appear in the exact posterior, however, we do not observe them in the approximated posterior; the shape of the approximation remains very simple. This is of course due to the assumption of factorization. We notice

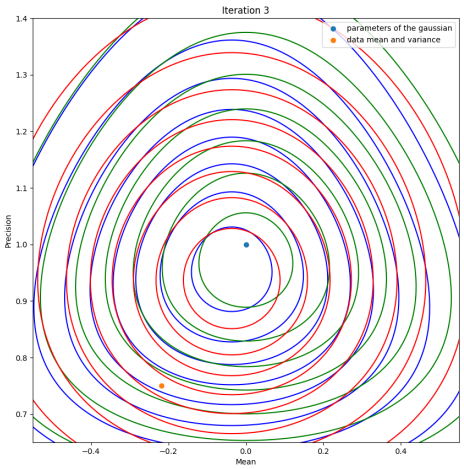

Figure 1: In green the prior; in blue the exact posterior; in red the approximated posterior. mean = 0, precision = 1, $a_0 = 20, b_0 = 20, \mu_0 = 0, \lambda_0 = 10, N\_points = 2$

however, that such tails disappear from the posterior as we increase the number of points (figure 5), in such case we get again a good approximation.

Lastly, in figure 6, we can observe a case where the posterior, the prior, and the true parameters are far from each other. Also in this case we notice that the exact posterior is a bit skewed and the approximation is not perfect. Moreover, the posterior is squeezed against the bottom axis. This is due to the particular values of a_0 and b_0.

In general the algorithm needs a very small number of iterations to converge.

The implementation of the algorithm is partially based on the code provided by MICHAEL P. J. CAMILLERI [2]

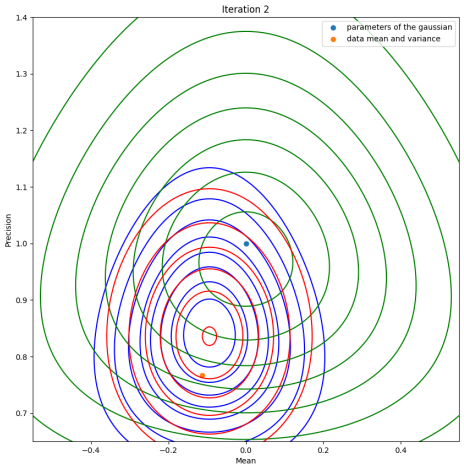

Figure 2: In green the prior; in blue the exact posterior; in red the approximated posterior. mean = 0, precision = 1, $a_0 = 20, b_0 = 20, \mu_0 = 0, \lambda_0 = 10, N\_points = 50$.

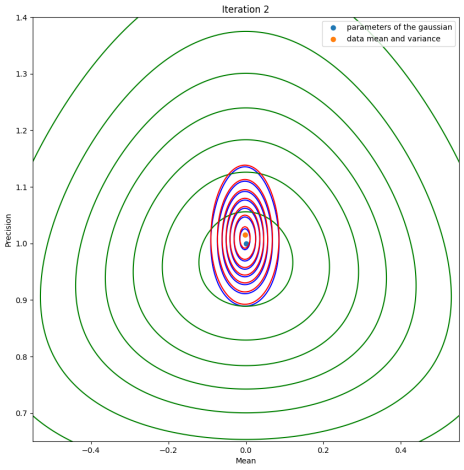

Figure 3: In green the prior; in blue the exact posterior; in red the approximated posterior. mean = 0, precision = 1, $a_0 = 20, b_0 = 20, \mu_0 = 0, \lambda_0 = 10, N\_points = 500$.

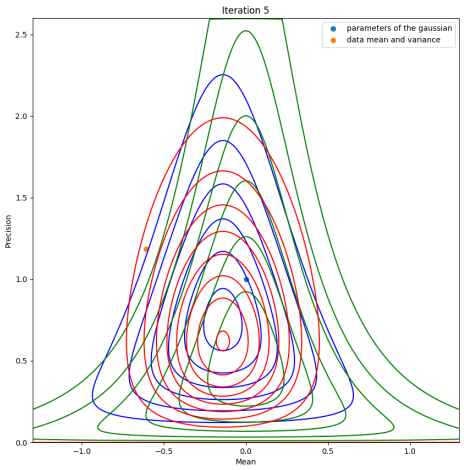

Figure 4: In green the prior; in blue the exact posterior; in red the approximated posterior.mean $= 0$, precision $= 1$, $\mathrm{a}_0 = 1, b_0 = 1, \mu_0 = 0, \lambda_0 = 10, N\_points = 3$.

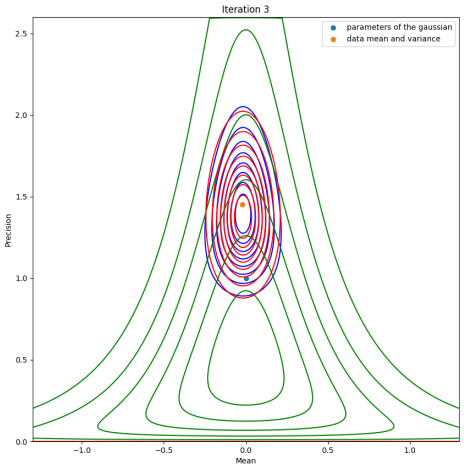

Figure 5: In green the prior; in blue the exact posterior; in red the approximated posterior. mean $= 0$, precision $= 1$, $\mathrm{a}_0 = 1, b_0 = 1, \mu_0 = 0, \lambda_0 = 10, N\_points = 50$.

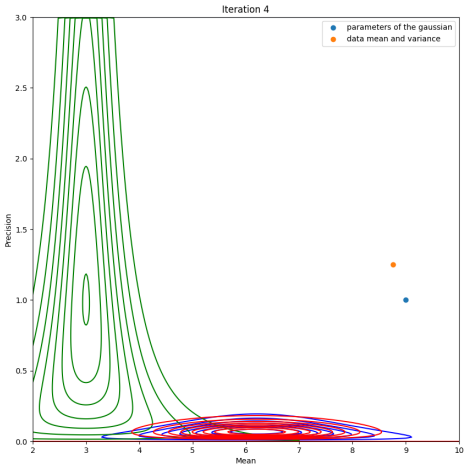

Figure 6: In green the prior; in blue the exact posterior; in red the approximated posterior. mean $= 9$, precision $= 1$, $\mathrm{a}_0 = 1, b_0 = 5, \mu_0 = 3, \lambda_0 = 4, N\_points = 5$.

2.4 Super Epicentra - Expectation-Maximization

Derivation of the algorithm

To derive the algorithm we will start from the general formulation of the EM algorithm.

Expectation step:

compute the responsibilities:

$$\gamma(Z_{ij}) = P(Z_j = 1|D_i, \vec{\Theta}^{old}) \tag{20}$$

for each $j \in 1, .., K$ and $i \in 1, .., N$; where K is the number of distributions in the mixture and N is the number of data points; Z_j is the jth component of the vector $\mathbf{Z}$ that is equal to one when a data point $D_i \in \mathbf{D}$ is generated by the jth distribution of the mixture.

Maximization step:

Maximize the log likelihood:

$$log(P(\mathbf{D}|\vec{\Theta})) = \sum_i^N \sum_j^K \gamma(Z_{ij}) log(P(D_i, Z_j|\vec{\Theta})) \tag{21}$$

In our case, $\mathbf{D} = (\mathbf{X}, \mathbf{S})$, where $\vec{x_i}$ and s_i are drawn respectively from a Gaussian and a Poisson distribution. Therefore in the expectation step we get:

$$\begin{aligned}
\gamma(Z_{ij}) &= P(Z_j = 1|s_i, \vec{x_i}, \vec{\Theta}^{old}) \\
&= \frac{P(Z_j = 1)P(s_i, \vec{x_i}|Z_j = 1, \vec{\Theta}^{old})}{\sum_l^k P(Z_l = 1)P(\vec{x_i}, s_i|Z_l = 1, \vec{\Theta}^{old})} \\
&= \frac{P(Z_j = 1)P(\vec{x_i}|Z_j = 1, \vec{\Theta}^{old})P(s_i|Z_j = 1, \vec{\Theta}^{old})}{\sum_l^k P(Z_l = 1)P(\vec{x_i}|Z_l = 1, \vec{\Theta}^{old})P(s_i|Z_l = 1, \vec{\Theta}^{old})} \\
&= \frac{\pi_j N(\vec{x_i}|\vec{u_j}, \mathbf{\Sigma}_j) Pois(s_i|\lambda_j)}{\sum_l^k \pi_l N(\vec{x_i}|\vec{u_l}, \mathbf{\Sigma}_l) Pois(s_i|\lambda_l)}
\end{aligned} \tag{22}$$

Where we used the Bayes formula, and we splitted the probability using the conditional indipendence given Z_j. Moreover, we kept the vector $\vec{\Theta}^{old}$ on the right side of the probability when we used the Bayes formula since it is just a constant. Lastly, we named πj the probability that $Z_j = 1$, therefore, the vector π represents the weights of the mixture.

Let's proceed now with maximization step:

$$
\begin{aligned}
log(P(\mathbf{D}|\vec{\Theta})) &= log(P(\mathbf{X},\mathbf{S}|\vec{\Theta})) \\
&= \sum_{i}^{N}\sum_{j}^{K}\gamma(Z_{ij})log(P(\vec{x_i},s_i,Z_j=1|\vec{\Theta})) \\
&= \sum_{i}^{N}\sum_{j}^{K}\gamma(Z_{ij})log(P(\vec{x_i},s_i|Z_j=1,\vec{\Theta})P(Z_j=1|\vec{\Theta})) \\
&= \sum_{i}^{N}\sum_{j}^{K}\gamma(Z_{ij})log(P(\vec{x_i}|Z_j=1,\vec{\Theta})P(s_i|Z_j=1,\vec{\Theta})P(Z_j=1|\vec{\Theta})) \\
&= \sum_{i}^{N}\sum_{j}^{K}\gamma(Z_{ij})[log(P(\vec{x_i}|Z_j=1,\vec{\Theta}))+log(P(s_i|Z_j=1,\vec{\Theta}))+log(P(Z_j=1|\vec{\Theta}))]
\end{aligned}
\tag{23}
$$

Where again we used the conditional independence to split the probabilities. To maximize this quantity we have to take the derivative with respect to each component of $\vec{\Theta}$ and set them equal to zero. Remember that the vector $\vec{\Theta}$ contains the parameters $\vec{\mu_l}$, $\mathbf{\Sigma}_l$, λ_l, and π_l for each $l \in 1,..,K$. Moreover, only the lth term of the sum over K depends on μ_l, $\mathbf{\Sigma}_l$, λ_l, and π_l. this helps us to simplify the derivatives.

Let's start from the derivative with respect to $\vec{\mu_l}$:

$$
\begin{aligned}
\frac{\partial log(P(\mathbf{X},\mathbf{S}))}{\partial\vec{\mu_l}} &= \sum_{i}^{N}\gamma(z_{il})\frac{\partial log(P(\vec{x_i}|Z_l=1,\vec{\mu_l},\mathbf{\Sigma}_l))}{\partial\vec{\mu_l}} \\
&= \sum_{i}^{N}\gamma(z_{il})\frac{\partial[log(\frac{1}{\sqrt{(2\pi)^2|\mathbf{\Sigma_l}|}})-0.5(\vec{x_i}-\vec{u_l})^T\mathbf{\Sigma}_l^{-1}(\vec{x_i}-\vec{u_l})]}{\partial\vec{\mu_l}} \\
&= \sum_{i}^{N}\gamma(z_{il})(\vec{x_i}-\vec{u_l})^T\mathbf{\Sigma}_l^{-1}
\end{aligned}
\tag{24}
$$

Where we used the the formula of the derivative of a quadratic form with respect to the vector, and the chain rule. Setting equation 24 equal to zero:

$$
\sum_{i}^{N}\gamma(z_{il})(\vec{x_i}-\vec{u_l})^T\mathbf{\Sigma}_l^{-1}=0 \tag{25}
$$

$$
\sum_{i}^{N}\gamma(z_{il})(\vec{x_i}-\vec{u_l})^T=0 \tag{26}
$$

$$\sum_{i}^{N} \gamma(z_{il})\vec{x_i} - \sum_{i}^{N} \gamma(z_{il})\vec{u_l} = 0 \tag{27}$$

$$\vec{u_l} = \frac{\sum_{i}^{N} \gamma(z_{il})\vec{x_i}}{\sum_{i}^{N} \gamma(z_{il})} \tag{28}$$

Now we can take the derivative with respect to $\mathbf{\Sigma}_l$, we are supposing that such matrix is diagonal $\mathbf{\Sigma}_l = \text{diag}(\sigma_{11}^l, \sigma_{22}^l)$, therefore, to avoid working with matrices we can take the derivatives with respect to the two elements of the diagonal; if we consider that $\vec{\mu_l} = (\mu_{l1}, \mu_{l2})$ and $\vec{x_i} = (x_{i1}, x_{i2})$, we have:

$$\begin{aligned}\frac{\partial log(P(\mathbf{X}, \mathbf{S}))}{\partial \sigma_{11}^l} &= \sum_{i}^{N} \gamma(z_{il}) \frac{\partial log(P(x_i | Z_l = 1, \vec{\mu_l}, \mathbf{\Sigma}_l))}{\partial \sigma_{11}^l} \\ &= \sum_{i}^{N} \gamma(z_{il}) [\frac{\partial log(\frac{1}{\sqrt{(2\pi)^2 \sigma_{11}^l \sigma_{22}^l}})}{\partial \sigma_{11}^l} - \frac{\partial 0.5((x_{i1} - u_{l1})^2 \frac{1}{\sigma_{11}^l} + (x_{i2} - u_{l2})^2 \frac{1}{\sigma_{22}^l})}{\partial \sigma_{11}^l}]\end{aligned} \tag{29}$$

where we calculated the determinant of the matrix, the inverse, and we developed the quadratic form. Considering that:

$$\begin{aligned}\frac{\partial log(\frac{1}{\sqrt{(2\pi)^2 \sigma_{11}^l \sigma_{22}^l}})}{\partial \sigma_{11}^l} &= \frac{\partial [-log(\sqrt{(2\pi)^2 \sigma_{11}^l \sigma_{22}^l})}{\partial \sigma_{11}^l}] \\ &= -\frac{1}{\sqrt{(2\pi)^2 \sigma_{11}^l \sigma_{22}^l}} \frac{1}{2\sqrt{(2\pi)^2 \sigma_{11}^l \sigma_{22}^l}} (2\pi)^2 \sigma_{22}^l \\ &= -0.5 \frac{1}{\sigma_{11}^l}\end{aligned} \tag{30}$$

moreover,

$$\frac{\partial 0.5((x_{i1} - u_{l1})^2 \frac{1}{\sigma_{11}^l} + (x_{i2} - u_{l2})^2 \frac{1}{\sigma_{22}^l})}{\partial \sigma_{11}^l} = -0.5(x_{i1} - u_{l1})^2 \frac{1}{(\sigma_{11}^l)^2} \tag{31}$$

joining equations 29, 30, 31 and setting the derivative equal to zero:

$$\frac{\partial log(P(\mathbf{X}, \mathbf{S}))}{\partial \sigma_{11}^l} = \sum_{i}^{N} -0.5\gamma(z_{il}) [\frac{1}{\sigma_{11}^l} - \frac{1}{(\sigma_{11}^l)^2} (x_{i1} - u_{l1})^2] = 0 \tag{32}$$

multiplying by σ_{11}^l:

$$\sum_{i}^{N} \gamma(z_{il})[1 - \frac{1}{\sigma_{11}^{l}}(x_{i1} - u_{l1})^2] = 0 \tag{33}$$

and we can conclude that:

$$\sigma_{11}^{l} = \frac{\sum_{i}^{N} \gamma(z_{il})(x_{i1} - u_{l1})^2}{\sum_{i}^{N} \gamma(z_{il})} \tag{34}$$

following the same steps we can also find that:

$$\sigma_{22}^{l} = \frac{\sum_{i}^{N} \gamma(z_{il})(x_{i2} - u_{l2})^2}{\sum_{i}^{N} \gamma(z_{il})} \tag{35}$$

Let's proceed now with the derivative with respect to λ_l

$$\begin{aligned} \frac{\partial log(P(\mathbf{X}, \mathbf{S}))}{\partial \lambda_l} &= \sum_{i}^{N} \gamma(z_{il}) \frac{\partial log(P(s_i | Z_l = 1, \lambda_l))}{\partial \lambda_l} \\ &= \sum_{i}^{N} \gamma(z_{il}) [\frac{\partial log(\lambda_l^{s_i}) + log(e^{-\lambda_l}) - log(s_i!)]}{\partial \lambda_l} \\ &= \sum_{i}^{N} \gamma(z_{il}) [\frac{\partial s_i log(\lambda_l) - \lambda_l - log(s_i!)]}{\partial \lambda_l} \\ &= \sum_{i}^{N} \gamma(z_{il}) [\frac{s_i}{\lambda_l} - 1] \end{aligned} \tag{36}$$

setting it equal to zero:

$$\sum_{i}^{N} \gamma(z_{il}) [\frac{s_i}{\lambda_l} - 1] = 0 \tag{37}$$

$$\lambda_l = \frac{\sum_{i}^{N} \gamma(z_{il}) s_i}{\sum_{i}^{N} \gamma(z_{il})} \tag{38}$$

Lastly, we compute the derivative with respect to π_l; here, we have to take in consideration that $\sum_{j}^{K} \pi_j = 1$; we will set this constraint using a Lagrange multiplier β:

$$\frac{\partial log(P(\mathbf{X},\mathbf{S}))}{\partial \pi_l} = \sum_i^N \gamma(z_{il}) \frac{\partial log(P(Z_l = 1|\pi_l)) + \beta(\sum_j^K \pi_j - 1)}{\partial \pi_l} \tag{39}$$

$$= \sum_i^N \gamma(z_{il}) \frac{1}{\pi_l} + \beta$$

$$\sum_i^N \gamma(z_{il}) \frac{1}{\pi_l} + \beta = 0 \tag{40}$$

$$\pi_l = -\frac{\sum_i^N \gamma(z_{il})}{\beta} \tag{41}$$

using the constraint we can calculate the value of the multiplier:

$$\sum_j^K \pi_j = 1 = \sum_j^K -\frac{\sum_i^N \gamma(z_{ij})}{\beta} \tag{42}$$

$$\beta = -\sum_j^K \sum_i^N \gamma(z_{ij}) = -N \tag{43}$$

substituting equation 43 in 41:

$$\pi_l = \frac{\sum_i^N \gamma(z_{il})}{N} \tag{44}$$

Implementation and results

To implement the algorithm, we first initialized the vector $\vec{\pi}$ uniformly, moreover, we randomly initialized the parameters λ_l and $\vec{\mu_l}$; lastly, we set the diagonal elements of the covariance matrices equal to the variances of the data in the two axis.

Plotting the data, we clearly observe three clusters; therefore, we would expect to get a good result running the algorithm with the parameter $k = 3$. Figure 7 and figure 8 show the position of the distributions at the first and at the last iteration respectively. As we can observe we got a very good result. However, we have to keep in mind that the EM algorithm converges to a local maximum, and therefore, depends on the initial positions of the distributions. Figure 9 illustrates the final position of the distributions if we start with a different initial condition. In this case, we did not get a good result. In general for a gaussian mixture, a good idea is to initialize the means of the gaussians equal to the centroid returned by the k-means algorithm. Moreover, we could run the algorithm many times any keep the solution that maximizes the log likelihood.

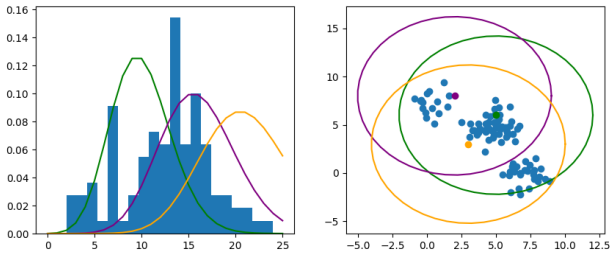

Figure 7: On the left the strengths of the earthquakes plotted using an histogram and the Poisson mixture; on the right, the locations of the earthquakes and the gaussian mixture, the ellipses represent the confidence intervals (3 times the standard deviation). Iteration number = 1; k = 3.

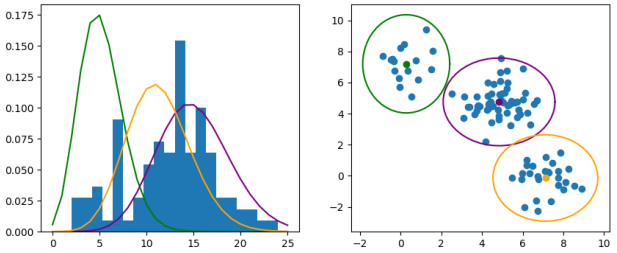

Figure 8: On the left the strengths of the earthquakes plotted using an histogram and the Poisson mixture; on the right, the locations of the earthquakes and the gaussian mixture, the ellipses represent the confidence intervals (3 times the standard deviation). Iteration number = 12; k = 3.

In figure 10 we plot the final position of the distributions when $k = 2$. As expected, one of the gaussians is covering two clusters.

Let's see now what happens if we try to set $k = 5$. In figure 11 we can notice a very interesting result; two of the Gaussians have a very low variance and they are concentrated over a very small number of points. In general, we will observe this kind of behavior when we use more gaussians than we need. This happens because when the mean of a gaussian is placed very close to a data point, the log likelihood becomes indefinitely high as the variance goes to zero. This is a very easy way to get a large likelihood, however, it does not mean that we got a good result. For instance, if we have a number of gaussians equal to the number of data points, we will observe that each gaussian will collapse on one of the data points and we would get a very high likelihood, this result however, is not useful. Moreover, this behavior leads to many complications in the implementation of the algorithm. First, if the mean of a gaussian is placed on a data point, we will

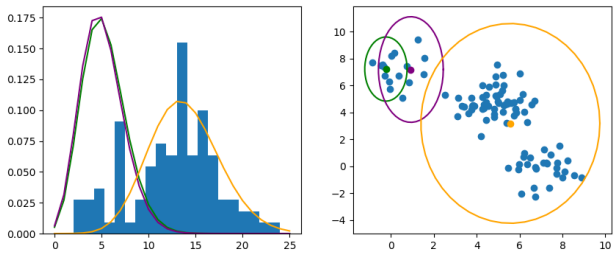

Figure 9: On the left the strengths of the earthquakes plotted using an histogram and the Poisson mixture; on the right, the locations of the earthquakes and the gaussian mixture, the ellipses represent the confidence intervals (3 times the standard deviation). Iteration number = 15; k = 3.

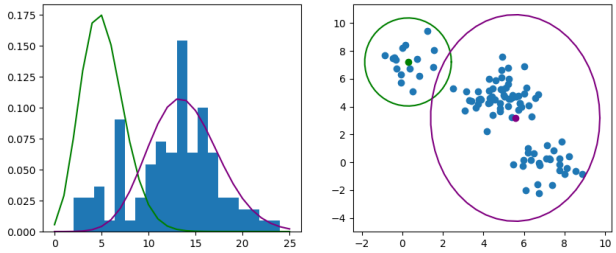

Figure 10: On the left the strengths of the earthquakes plotted using an histogram and the Poisson mixture; on the right, the locations of the earthquakes and the gaussian mixture, the ellipses represent the confidence intervals (3 times the standard deviation). Iteration number = 8; k = 2.

get a $\mathbf{0}$ covariance matrix, and since we have the determinant of this matrix in the denominator of the gaussian pdf, we cannot compute the likelihood. Second, even if the mean is not in the same position of a data point, when the variances of the gaussian are very low, Python will approximate the likelihood that a data point was generated by that gaussian to zero when such data point is far from the mean. This approximation does not allow us to compute the log likelihood to check for the convergence of the algorithm. In fact, in the formula of the log likelihood of the data we will have a $log(0)$. To solve these problems we can check if the covariance matrix becomes equal to the zero matrix, and in such case, we can reinitialize the mean and the covariance matrix to some new values. This in general could be done whenever the variances get very low, checking if the likelihood of the points are apporoximated to zero. What I observed however, is that in most of cases even if we reinitialize a gaussian, it will tend to collapse on the same data points again. We have to remember, that in our problem

we are fitting some gaussians and some poissons at the same time, therefore, we could observe different behaviours if we work just with gaussians. Notice that we never observe a rotation of the ellipses, this is beacuse we are using a diagonal covariance matrix.

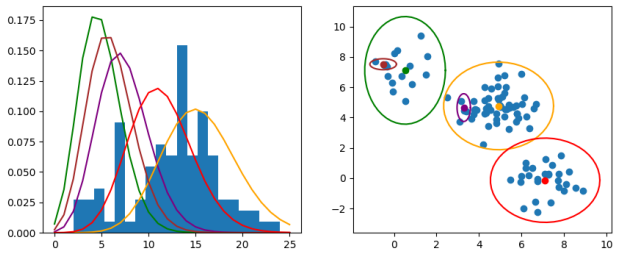

Figure 11: On the left the strengths of the earthquakes plotted using an histogram and the Poisson mixture; on the right, the locations of the earthquakes and the gaussian mixture, the ellipses represent the confidence intervals (3 times the standard deviation). Iteration number = 8; k = 5.

Figure 12 illustrates the log likelihood at each iteration for different values of k. As expected, we get the maximum value when $k = 3$. We know that the EM algorithm increases the log likelihood at each step, however, in the plot we can observe that sometimes it decreases, this due to the reinitialization of the gaussian; every time a gaussian is reinitialized, the log likelihood slightly decreases, moreover, after the reinitialization we never observe a significant improvement in the log likelihood.

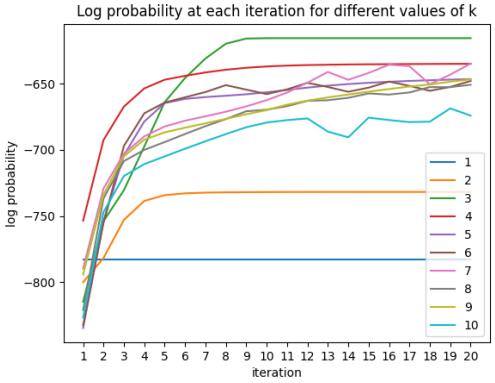

Figure 12: The log likelihood for different values of K at each iteration. The likelihood decreases when a gaussian is reinitialized.

It is also worth to mention that the problem of the collapsing guassian could be

solved using a bayesian approach; i.e, we consider a prior distribution for the parameters where we have a low probability to get low variances. Using MAP rather than MLE, it is less likely that we will observe the phenomenon of the collapsing gaussian.

To conclude, we provide the values of the π vector when $k = 3$:

0.2700935758292452

0.16000900078909946

0.5698974233816554

We can observe that they sum up to one, and one has a significantly higher value .

References

[1] R. C. Steorts and L. Qian, "Introduction to the normal gamma model."

[2] M. P. J. CAMILLERI, "Variational inference for bayesian univariate gaussian."

Dynamic algorithm

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

map = {}

def S(node_u, value_k,tree_topology, theta, beta,values = [0,1,2,3,4]):
    # print(k)
    children = []
    for i in range(len(tree_topology)):
        if tree_topology[i] == node_u:
            children.append(i)
    if not children:
        return -1
    product = 1
    for child in children:
        sum = 0
        for value in values:
            key = str(child)+" "+str(value)
            s = map.get(key)
            if s is None:
                s = S(child, value, tree_topology, theta, beta)
                map[key] = s
            if s == -1:

                sum += theta[child][value_k][int(beta[child])]
                break
            else:
                sum += s*theta[child][value_k][value]
        product *= sum
    return product


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):
    :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.
    This is a suggested template. You don't have to use it.
    """
    sum = 0
    values = [0, 1, 2, 3, 4]
```

```
    for value in values:
        sum += S(0,value,tree_topology,theta,beta)*theta[0][value]

    # Start: Example Code Segment. Delete this segment completely before you implement the c
    print("\tCalculating the likelihood...")
    likelihood = sum
    # End: Example Code Segment
    for k in list(map):
        del map[k]
    return likelihood


def main():
    print("Hello World!")
    print("This file is the solution template for question 2.2.")

    print("\n1. Load tree data from file and print it\n")

    filename = "data/q2_2_small_tree.pkl"  # "data/q2_2_medium_tree.pkl", "data/q2_2_large_t
    print("filename: ", filename)

    t = Tree()
    t.load_tree(filename)
    t.print()
    print("K of the tree: ", t.k, "\talphabet: ", np.arange(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("\n\tSample: ", sample_idx, "\tBeta: ", beta)
        sample_likelihood = calculate_likelihood(t.get_topology_array(), t.get_theta_array()
        print("\tLikelihood: ", sample_likelihood)


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

Variational Inference algorithm

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

tolerance = 1e-4
#parameters of the normal that generates the points
mean = 9
precision = 1
#parameters of the prior distribution
#gamma
a_0 = 1
b_0 = 0.5
#gaussian
mu_0 = 3
lambda_0 = 4
#number of points
N_points = 5
#compute the prior for some values of the mean and the precision
mean_axis = np.linspace(2, 10, 400)
tau_axis = np.linspace(0, 3, 400)

prior_values = np.empty([len(mean_axis), len(tau_axis)])
p_tau = gamma(a=a_0, loc=0, scale=1/b_0)
for i, tau in enumerate(tau_axis[1:]):
    p_mu = norm(loc=mu_0, scale=1/np.sqrt(lambda_0 * tau))
    prior_values[i,:] = p_tau.pdf(tau) *  p_mu.pdf(mean_axis)

# generate points
data = np.random.normal(loc=mean, scale=np.sqrt(1 / precision), size=N_points)

#compute exact posterior
#it is interesting to note that the parameters depend on the number of points
mu_N = (lambda_0 * mu_0 + np.sum(data))/(N_points + lambda_0)
a_N = a_0 + N_points*0.5
b_N = b_0 + 0.5 *
      lambda_0 * (mu_0**2) + 0.5 * np.square(data).sum() - 0.5
      * (N_points + lambda_0) * mu_N**2

exact_posterior_values = np.empty([len(mean_axis), len(tau_axis)])
p_tau = gamma(a=a_N, loc=0, scale=1/b_N)
for i, tau in enumerate(tau_axis[1:]):
    p_mu = norm(loc=mu_N, scale=1/np.sqrt((N_points+lambda_0) * tau))
    exact_posterior_values[i,:] = p_mu.pdf(mean_axis) * p_tau.pdf(tau)

```

```
45
46  E_tau = 10
47  E_tau_prev = np.NaN
48
49  iteration = 0
50  while np.isnan(E_tau_prev) or (np.abs(E_tau - E_tau_prev) >
51  tolerance and iteration < 10):
52      mu_N = (lambda_0 * mu_0 + N_points * data.mean()) / (lambda_0 + N_points)
53      lambda_N = (lambda_0 + N_points) * E_tau
54      E_mu = mu_N
55      E_mu2 = lambda_N ** -1 + E_mu ** 2
56      a_N = a_0 + (N_points  + 1) / 2
57      b_N = b_0 + 0.5 * (np.square(data) - 2
58            * E_mu * data + E_mu2).sum() + 0.5 * lambda_0 *
59            (E_mu2 - 2 * E_mu * mu_0 + mu_0 ** 2)
60      E_tau_prev = E_tau
61      E_tau = a_N / b_N
62
63      pTau = gamma(a=a_N, loc=0, scale=1 / b_N)
64      pMu = norm(loc=mu_N, scale=1 / np.sqrt(lambda_N))
65
66      values = np.outer(pMu.pdf(mean_axis), pTau.pdf(tau_axis))
67
68      # Now Plot
69      plt.figure(figsize=[10, 10])
70      plt.plot(mean, precision, 'o', label='parameters of the gaussian')
71      plt.plot(data.mean(), 1 / data.var(), 'o', label='data mean and variance')
72      plt.contour(*np.meshgrid(mean_axis, tau_axis), exact_posterior_values,
73                  levels=8, colors=['blue'])
74      plt.contour(*np.meshgrid(mean_axis, tau_axis), prior_values, levels=8, colors=['green'])
75      plt.contour(*np.meshgrid(mean_axis, tau_axis), values.T, levels=8, colors=['red'])
76
77      plt.title('Iteration {}'.format(iteration))
78      plt.xlabel('Mean')
79      plt.ylabel('Precision')
80      plt.legend(loc="upper right")
81      plt.show()
82      iteration += 1
```

EM algorithm

```
import numpy as np, numpy.random
from scipy.stats import multivariate_normal as normal
from scipy.stats import poisson
import matplotlib.pyplot as plt
import random
import matplotlib as mpl
from matplotlib.patches import Ellipse

def column(matrix, i):
    return [row[i] for row in matrix]


def EM (S,X, n, k):

    X0 = column(X, 0)
    X1 = column(X, 1)
    tolerance = 0.001
    prob = np.zeros(n)
    pi = [1/k for i in range(k)]
    responsibility = np.zeros((k,len(S)))
    lambda_ = [random.randrange(int(np.min(S)),int(np.max(S))) for i in range(k)]
    mu = [[random.randrange(int(np.min(X0)),int(np.max(X0))),
           random.randrange(int(np.min(X1)),int(np.max(X1)))] for i in range(k)]
    sigma = np.ones((k,2,2))
    for i in range(k):
        sigma[i] = [[np.var(X0),0],[0,np.var(X1)]]



    fig = plt.figure(figsize=(10,4))
    ax = fig.add_subplot(121)
    ax2 = fig.add_subplot(122)
    #plot poisson
    colors = ["green","purple","orange","brown","red","yellow","pink"]
    ax.hist(S, bins=20, density=True)
    for j in range(k):
        x = [i for i in range(0, 26)]
        y = [poisson.pmf(i, lambda_[j]) for i in range(0, 26)]
        ax.plot(x, y, color=colors[j])
    #plot gaussians
    ax2.scatter(X0, X1)
    for i in range(k):
        ax2.scatter(mu[i][0], mu[i][1], color=colors[i])
        u = mu[i][0]  # x-position of the center
```

```
        v = mu[i][1]  # y-position of the center
        a = np.sqrt(sigma[i][0][0]) * 3  # radius on the x-axis
        b = np.sqrt(sigma[i][1][1]) * 3  # radius on the y-axis
        t = np.linspace(0, 2 * np.pi, 50)
        ellipse_x = [u + a * np.cos(j) for j in t]
        ellipse_y = [v + b * np.sin(j) for j in t]
        ax2.plot(ellipse_x, ellipse_y, '-', color=colors[i])
    plt.savefig('images/test6/{}.png'.format(0))
    plt.show()



    iteration = 0
    while iteration < n and (iteration < 2 or
           np.abs(prob[iteration-1]-prob[iteration-2]) >= tolerance):
        #expectation
        for i in range(k):
            for j in range(len(X)):
                responsibility[i][j] = pi[i]*normal.pdf( X[j],mean=mu[i], cov=sigma[i] )
                *poisson.pmf(S[j], mu=lambda_[i])
                sum = 0
                for l in range(k):
                    sum +=pi[l]*normal.pdf(X[j],mean=mu[l],cov=sigma[l])*
                    poisson.pmf(S[j],lambda_[l])
                responsibility[i][j] /= sum
        #maximization
        for i in range(k):
            tot_responsibility = np.sum(responsibility[i])
            mu[i] = np.zeros(2)
            sigma[i] = np.zeros((2,2))
            lambda_[i] = 0
            for j in range(len(X)):
                mu[i] += X[j]*responsibility[i][j]
            mu[i] /= tot_responsibility
            for j in range(len(X)):
                lambda_[i] += S[j]*responsibility[i][j]
                sigma[i][0][0] += responsibility[i][j]* (X[j][0] - mu[i][0])**2
                sigma[i][1][1] += responsibility[i][j]* (X[j][1] - mu[i][1])**2
            sigma[i] /= tot_responsibility
            lambda_[i] /= tot_responsibility
            pi[i] = tot_responsibility/len(X)

            #check if the matrix is semidefinite positive
            if np.linalg.det(sigma[i]) == 0:
                sigma[i] = [[np.var(X0), 0], [0, np.var(X1)]]
                mu[i] = [random.randrange(int(np.min(X0)),
```

```
                        int(np.max(X0))),random.randrange(int(np.min(X1)),int(np.max(X1)))]
                # sigma[i] += 1.0e-8*numpy.identity(2)

        #compute log probability
        for j in range(len(S)):
            for i in range(k):
                poisson_p = poisson.pmf(S[j],lambda_[i])
                normal_p = normal.pdf(X[j],mean=mu[i],cov=sigma[i])
                if normal_p == 0:
                    sigma[i] = [[np.var(X0), 0], [0, np.var(X1)]]
                    mu[i] = [random.randrange(int(np.min(X0)),
                             int(np.max(X0))),random.randrange(int(np.min(X1)),
                             int(np.max(X1)))]
                    normal_p = normal.pdf(X[j], mean=mu[i], cov=sigma[i])
                prob[iteration] += responsibility[i][j] * (np.log(normal_p)
                                   +np.log(poisson_p) + np.log(pi[i]))
        iteration += 1


        fig = plt.figure(figsize=(10, 4))
        ax = fig.add_subplot(121)
        ax2 = fig.add_subplot(122)
        ax.hist(S, bins=20, density=True)
        for j in range(k):
            x = [i for i in range(0, 26)]
            y = [poisson.pmf(i, lambda_[j]) for i in range(0, 26)]
            ax.plot(x, y, color=colors[j])
        # plot gaussians
        ax2.scatter(X0, X1)
        for i in range(k):
            ax2.scatter(mu[i][0], mu[i][1], color=colors[i])
            u = mu[i][0]  # x-position of the center
            v = mu[i][1]  # y-position of the center
            a = np.sqrt(sigma[i][0][0]) * 3  # radius on the x-axis
            b = np.sqrt(sigma[i][1][1]) * 3  # radius on the y-axis
            t = np.linspace(0, 2 * np.pi, 50)
            ellipse_x = [u + a * np.cos(j) for j in t]
            ellipse_y = [v + b * np.sin(j) for j in t]
            ax2.plot(ellipse_x, ellipse_y, '-', color=colors[i])
        plt.savefig('images/test6/{}.png'.format(iteration))
        plt.show()

    #plot logs
    # x = [i for i in range(1,n+1)]
    # plt.xticks(x)
    # plt.plot(x,prob[:], label=(k))
```

```
137     return pi, lambda_, mu, sigma, np.max(prob)
138
139 X = np.loadtxt("X.txt")
140 S = np.loadtxt("S.txt")
141
142
143 pi,lambda_,mu,sigma, prob = EM(S,X,50,2)
```

I discussed the assignment with my teammate Alice De Schutter.