# Assignment 2

Callum Lau
Probabilistic and Unsupervised Learning

November 12, 2019

**Problem 1.**

**Solution** Part a): The likelihood is given by:

$$L(\theta) = \prod_{N=1}^{N}\sum_{k=1}^{K}\pi_k P(\mathbf{x}_n;\theta_k) = \prod_{n=1}^{N}\left\{\sum_{k=1}^{K}\pi_k\prod_{d=1}^{D}\theta_{kd}^{x_{nd}}(1-\theta_{kd})^{1-x_{nd}}\right\}$$

Hence the log likelihood is given by:

$$l(\theta) = \sum_{n=1}^{N}\log\sum_{k=1}^{K}\pi_k P(\mathbf{x}_n;\theta_k) = \sum_{n=1}^{N}\log\left\{\sum_{k=1}^{K}\pi_k\prod_{d=1}^{D}\theta_{kd}^{x_{nd}}(1-\theta_{kd})^{1-x_{nd}}\right\},$$

where $x_{nd}$ is the $dth$ data point of the $nth$ sample, and $\theta_{kd} = Pr(x_d = 1)$ for the $kth$ mixture. Introducing the latent variable $s^{(n)} \in \{(1,...,K)\}$ such that $P(s^(n) = k|\pi) = \pi_k$, and associated indicator variable $\mathbf{s}$ such that $\mathbf{s}_i = k \iff \mathbf{s}_i = [0,0,...,1,...,0]$ with 1 at $kth$ position then we can rewrite mixture as a latent variable model using the formula for exponential family distributions. Note that:

$$\begin{aligned}
P(\mathbf{x}|\mathbf{p}) &= \exp\left(\log\prod_{d=1}^{D}\theta_d^{x_d}(1-\theta_d)^{(1-x_d)})\right)\\
&= \exp\left(\sum_{d=1}^{D}\log(1-\theta_d) + \sum_{d=1}^{D}x_d\log(\frac{\theta_d}{1-\theta_d})\right)\\
&= \prod_{d=1}^{D}(1-\theta_d)\exp\left(\begin{bmatrix}\log(\frac{\theta_1}{1-\theta_1})\\ \vdots \\ \log(\frac{\theta_D}{1-\theta_D})\end{bmatrix}^T\cdot\begin{bmatrix}x_1\\ \vdots\\ x_D\end{bmatrix}\right)
\end{aligned}$$

hence $g(\theta) = \prod_{d=1}^{D}(1-\theta_d)$, $\theta = \begin{bmatrix}\log(\frac{\theta_1}{1-\theta_1})\\ \vdots \\ \log(\frac{\theta_D}{1-\theta_D})\end{bmatrix}$ and $T(\mathbf{x}) = \begin{bmatrix}x_1\\ \vdots\\ x_D\end{bmatrix}$ and collecting the K component distributions' natural parameters in the $(K\times D)$ matrix $\mathbf{P}^T = \begin{pmatrix}\log(\frac{\theta_{11}}{1-\theta_{11}}) & \dots & \log(\frac{\theta_{1D}}{1-\theta_{1D}})\\ \vdots & \vdots & \vdots\\ \log(\frac{\theta_{K1}}{1-\theta_{K1}}) & \dots & \log(\frac{\theta_{KD}}{1-\theta_{KD}})\end{pmatrix}$

We can use the formula for the joint-data-log-likelihood:

$$\begin{aligned}
\log P(X,S) &= \sum_{n=1}^{N} \left\{ (\log \boldsymbol{\pi})^T \mathbf{s}_n + \mathbf{s}_n^T \mathbf{P}^T T(\mathbf{x}_n) + \mathbf{s}_n^T \log \mathbf{g}(\boldsymbol{\Theta}) \right\} \\
&= \sum_{n=1}^{N} \Bigg( \begin{pmatrix} \log(\pi_1) & \dots & \log(\pi_K) \end{pmatrix} \cdot \mathbf{s}_n + \mathbf{s}_n^T \begin{pmatrix} \log(\frac{\theta_{11}}{1-\theta_{11}}) & \dots & \log(\frac{\theta_{1D}}{1-\theta_{1D}}) \\ \vdots & \vdots & \vdots \\ \log(\frac{\theta_{K1}}{1-\theta_{K1}}) & \dots & \log(\frac{\theta_{KD}}{1-\theta_{KD}}) \end{pmatrix} \begin{pmatrix} x_{n1} \\ \vdots \\ x_{nD} \end{pmatrix} \\
&\quad + \mathbf{s}_n^T \begin{pmatrix} \sum_{d=1}^{D} \log(1-\theta_{1d}) \\ \vdots \\ \sum_{d=1}^{D} \log(1-\theta_{Kd}) \end{pmatrix} \Bigg) \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} s_{nk} \left\{ \log \pi_k + \sum_{d=1}^{D} x_{nd} \log \theta_{kd} + (1-x_{nd}) \log(1-\theta_{kd}) \right\} = (*)
\end{aligned}$$

Part b) Note that:

$$r_{nk} = P(s^{(n)} = k | \mathbf{x}_n, \boldsymbol{\pi}, \mathbf{P}) = P(s_{nk} = 1 | \mathbf{x}_n, \boldsymbol{\pi}, \mathbf{P})$$

$$\begin{aligned}
&\overset{Bayes}{=} \frac{P(\mathbf{x}_n | s_{nk} = 1, \mathbf{P}, \boldsymbol{\pi}) P(s_{nk} = 1 | \boldsymbol{\pi}, \mathbf{P})}{P(\mathbf{x}_n)} \\
&= \frac{\pi_k \prod_{d=1}^{D} \theta_{kd}^{x_{nd}} (1-\theta_{kd})^{(1-x_{nd})}}{\sum_{j=1}^{K} P(\mathbf{x}_n | \mathbf{P}) \pi_j} \\
&= \frac{\pi_k \prod_{d=1}^{D} \theta_{kd}^{x_{nd}} (1-\theta_{kd})^{(1-x_{nd})}}{\sum_{j=1}^{K} \pi_j \prod_{d=1}^{D} \theta_{jd}^{x_{nd}} (1-\theta_{jd})^{1-x_{nd}}}
\end{aligned}$$

Part c)

$$\begin{aligned}
\Big\langle \sum_{n=1}^{N} \log P(\mathbf{x}_n, s^{(n)} | \boldsymbol{\pi}, \mathbf{P}) \Big\rangle_q &= \langle (*) \rangle \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbb{E}(s_{nk}) \left\{ \log \pi_k + \sum_{d=1}^{D} x_{nd} \log \theta_{kd} + (1-x_{nd}) \log(1-\theta_{kd}) \right\} \\
&= \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \left\{ \log \pi_k + \sum_{d=1}^{D} x_{nd} \log \theta_{kd} + (1-x_{nd}) \log(1-\theta_{kd}) \right\}
\end{aligned}$$

Then to find the maximising $\mathbf{P}$ parameter, differentiate w.r.t $\theta_{ij}$:

$$\frac{\partial \langle (*) \rangle}{\partial \theta_{ij}} = \sum_{n=1}^{N} r_{ni} \left\{ \frac{x_{nj}}{\theta_{ij}} - \frac{1-x_{nj}}{1-\theta_{ij}} \right\} \overset{!}{=} 0$$

$$\implies \hat{\theta}_{ij} \sum_{n=1}^{N} r_{ni} = \sum_{n=1}^{N} r_{ni} x_{nj}$$

$$\hat{\theta}_{ij} = \frac{\sum_{n=1}^{N} r_{ni} x_{nj}}{\sum_{n=1}^{N} r_{ni}}$$

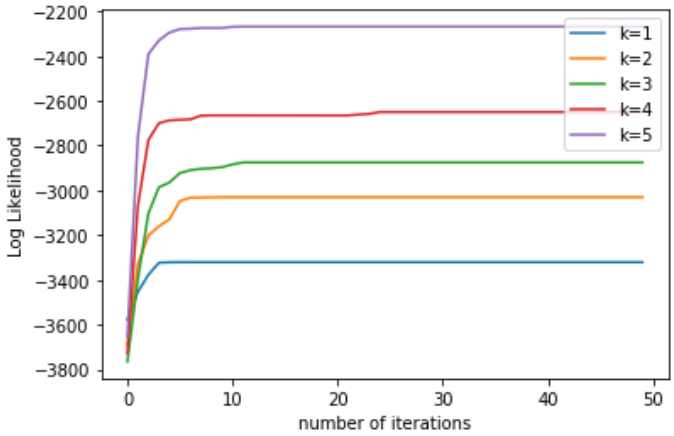
To find maximising $\boldsymbol{\pi}$ parameter, form the Lagrangian under the condition $\sum_{k=1}^{K} \pi_k = 1$:

$$L = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \left\{ \log \pi_k + \sum_{d=1}^{D} x_{nd} \log \theta_{kd} + (1 - x_{nd}) \log(1 - \theta_{kd}) \right\} + \lambda \left( \sum_{k=1}^{K} \pi_k - 1 \right)$$

$$\implies \frac{\partial L}{\partial \pi_m} = \sum_{n=1}^{N} \frac{r_{nm}}{\pi_m} + \lambda \stackrel{!}{=} 0 \implies \pi_m = -\frac{\sum_{n=1}^{N} r_n m}{\lambda} \implies \sum_{m=1}^{K} \pi_m = -\frac{\sum_{m=1}^{K} \sum_{n=1}^{N} r_{nm}}{\lambda}$$

$$\implies \lambda = -\sum_{m=1}^{K} \sum_{n=1}^{N} r_{nm} = N \implies \pi_m = \frac{\sum_{n=1}^{N} r_{nm}}{N}$$

Part d) See code attached - A description of the code is given in the header of the file.
Log-Likelihoods:




Posterior Probabilities for K= 2, 3 and 7 are given:

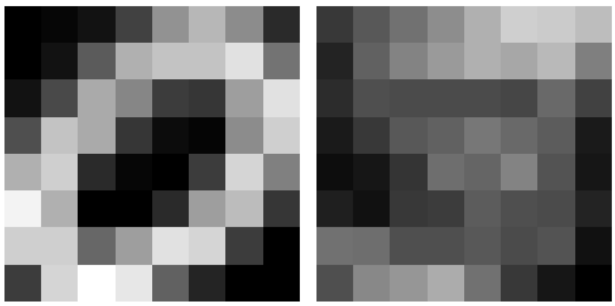


Figure 1: Posterior Probabilities K=2

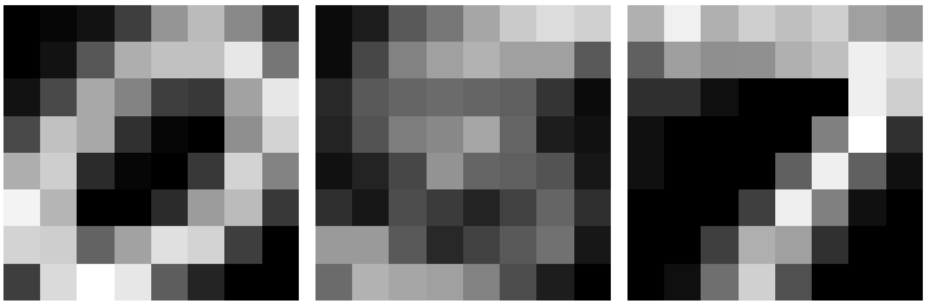


Figure 2: Posterior Probabilities K=3

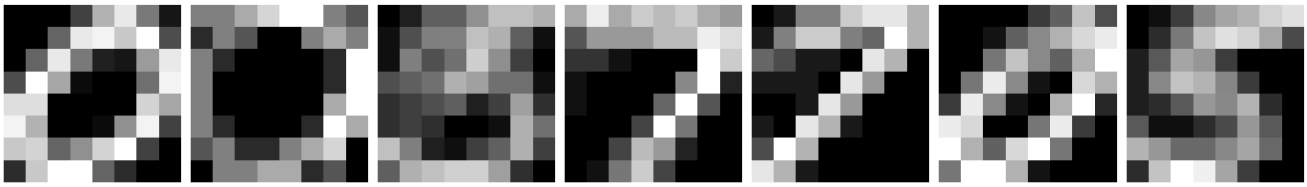


Figure 3: Posterior Probabilities K=7

Part e) For different values of K, the log likelihood expressed in bits was calculated. This number corresponds to the "information loss" or entropy of the algorithm. The naive encoding of the data stores every bit: N $\times$ D = 100 $\times$ 64 = 6400 bits in total. The bits per dim/pixel of the algorithm is therefore expressed as $\frac{\text{-(loglikehood in bits)}}{\#\text{(bits in naive encoding)}}$

| Log Likelihood in bits compared to length of naive encoding | | |
|---|---|---|
| K value | Log-likehood | Bits/Dim |
| 2 | -4766.375494092242 | $\frac{6400-4766}{6400} = 0.74$ |
| 3 | -4408.98036379874 | $\frac{6400-4409}{6400} = 0.69$ |
| 4 | -4114.073141290119 | $\frac{6400-4114}{6400} = 0.64$ |
| 7 | -3587.469009552203 | $\frac{6400-3587}{6400} = 0.56$ |
| 10 | -3250.1344257061214 | $\frac{6400-3250}{6400} = 0.51$ |

These numbers make sense: the more clusters used, the less average number of bits needed to compress all the pixels. Using gzip, the original binarydigits.txt file size (including whitespace) was 13kB, whereas the compressed file was 2kB, leading to a bits per pixel of roughly 0.15. The huge difference in ratio can be explained by the fact that for binary data, the information required to encode the model parameters is not efficient compared to the dimensionality of the data. Conversely gzip is designed specifically for binary data, and the pixel data is characterised by long strings of 0s and 1s, which can be stored extremely efficiently using a much smaller table system.

Part f) After producing the cluster means, each cluster was rearranged into an (8x8) image, and the resulting images analysed. Essentially the algorithm can be thought of as attempting to classify the N=100 images into K clusters. Note that from inspection, the

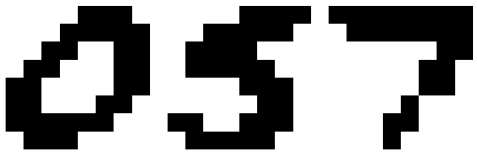
images are all clearly the digits 0, 5 and 7.

| Analysis of Solutions | |
|---|---|
| K value | Same solutions? (up to permutation) |
| 2 | No - generally the digit 0 was correctly classified by one cluster, however the next cluster produced an image, which was either a 5 or 7, or some combination of both. |
| 3 | To an extent yes - in every run, the images in the data set were correctly clustered into the digits 0, 5 and 7. The algorithm struggled however to classify the digit 5, and the clarity of the image produced varied between runs. |
| 4 | No - the digits 0,5 and 7 always appear at least once in one of the 4 clusters, however, for the remaining cluster, the digit produced resembled one of the three digits (different each time) or some combination of them. |
| 7 | No - similar result to 4. Each digit appears at least once, with generally 2 digits appearing twice. Again, the last cluster was some variation on 0,5 or/and 7 |
| 10 | No - similar result to 7. |

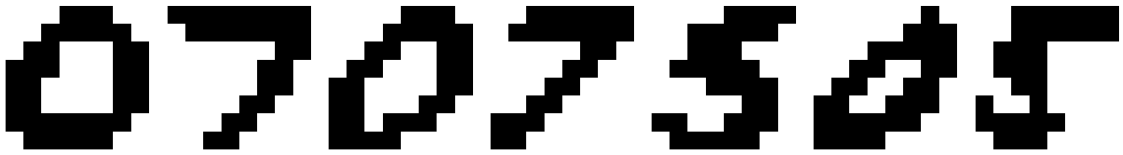
In conclusion, for K=3, the algorithm does produce very similar solutions with randomised starting conditions. However, this conclusion does depend on the value of K, as for the other K values, different solutions appear with different frequency. This is to be expected since we know that the data clearly consists of 3 clusters, and therefore the lack of, or excess of clusters will result in different solutions each time. The cluster images for different K value are produced below, where the pixel is classified as black or white, if the cluster value is high or lower than the cluster mean:

K = 2

K = 3



K = 7



In summary, the algorithm works well at finding the clusters, as for K=3, it correctly classifies the images into the three categories (most of the time), as shown by the images above. The algorithm does struggle to identify the 5 digit, however, this is well reflected by the responsibilities of the clusters. For example, on one run, the algorithm produced responsibilities of $[0.41, 0.41, 0.18]$ corresponding to the cluster displaying 0,7 and 5 respectively. On brief inspection of the training data, the digit 5 occurs 20% of the time, whereas the other 0 and 7 appear 40% of the time, accurately corresponding to the responsibilities the clusters take, and explaining why the algorithm has a harder time classifying the digit 5. The model may be improved therefore by providing more training images, particularly of 5s, so that it is better at differentiating between 5s and other digits.

Part (g)

For each pixel, we need to store the identity of the cluster it was assigned to, and since there are K clusters, this requires $log_2 K$ bits per pixel. We also need to store the K cluster centre vectors themselves, and therefore the total cost of encoding the parameters and data is:

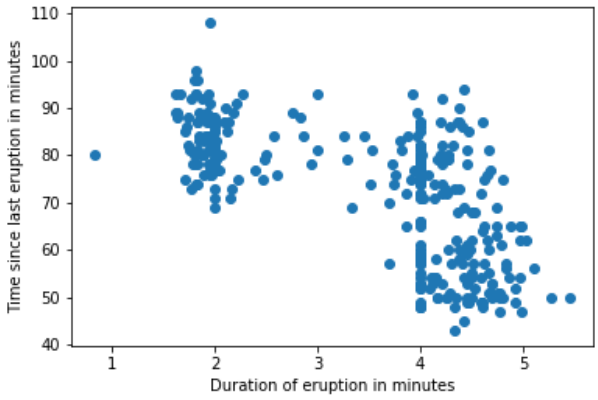
$$K + NDlog_2 K \tag{1}$$

Which for all K, is greater than 6400. Therefore using a mixture of Bernoulli for compression of binary data is not efficient at all. Comparatively, as shown before gzip compressed the data to 15% of the original file, and therefore is much better.

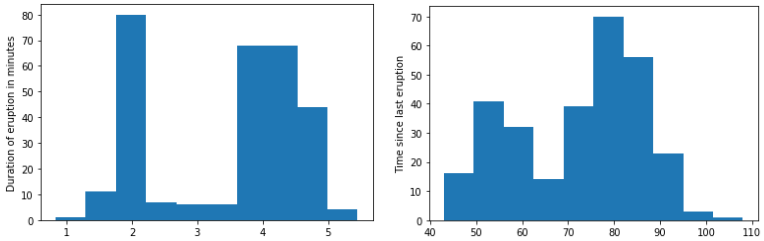
**Problem 2.**

**Solution** Part a)

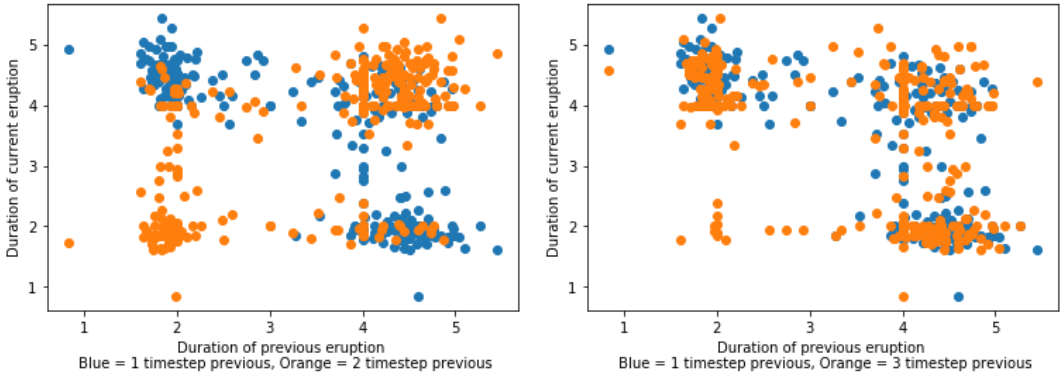
Observation 1 From the plot of the variables within timesteps, we see that the data exhibits strong time dependence between each timestep. This is seen via a sequential plot of the data points (Attached is a video timesteps.mpg4). The duration of eruption generally alternates at each time step between a long eruption time (4 minutes) and a short eruption time (2 minutes), suggesting that the current eruption length is dependent at least on the previous eruption length. From the plot below, it is clear that for high waiting times since last eruption ($> 70$ mins), the duration of the eruption is equally as likely to be short (2 mins) as long (4 mins), whereas for a short time since the last eruption ($< 70$ mins), the duration of the eruption is always long (4 mins).



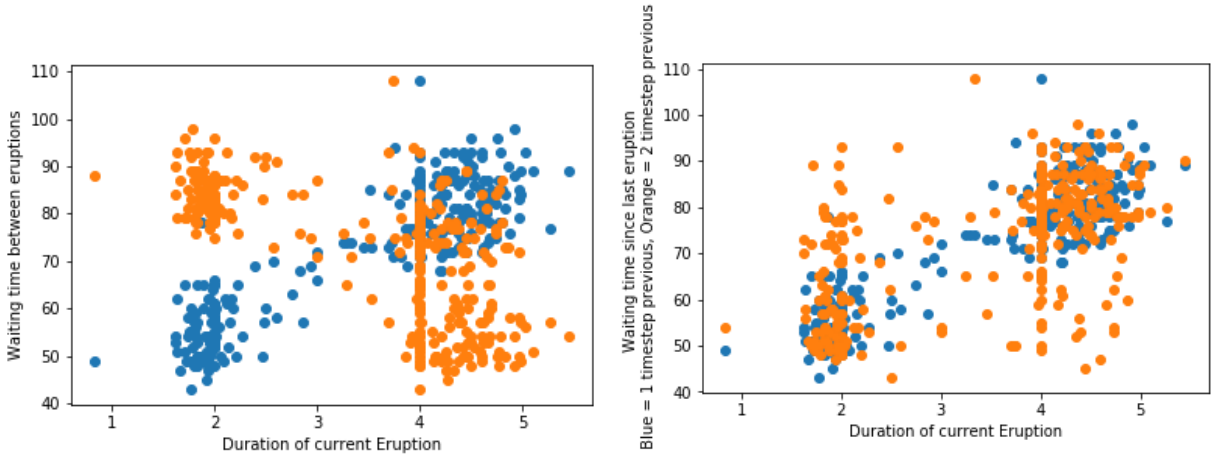

Observation 2 From two histogram plots of the individual variables, we see that both time since last eruption and eruption duration potentially follow a mixture of two Gaussian distributions:




Observation 3 Below are plots of Duration of Eruption vs Duration of Eruption between timesteps. For both plots, the blue points can be thought of as the duration of the 'current' eruption (y-axis) vs the duration of the previous eruption (x-axis). Note that for current short duration eruptions, the previous eruption is only ever long, whereas for current long duration eruptions, the previous eruption is long or short. This suggests that each eruption duration is conditionally dependent on the last.The orange points show the 'current' eruption duration vs the eruption duration either two (left diagram) or three (right diagram) steps back. These points are more uniformly spread out (around the 'square' of possible values) suggesting the dependence weakens over time.



Observation 4 Below are plots of the duration of the current eruption duration (x-axis)) vs waiting time till next eruption (y-axis) between timesteps. The blue points show the waiting time immediately after current eruption. These show a somewhat linear relationship - short duration eruptions 'result' in short waiting times, and long duration eruptions result in a long waiting time. The orange points show the waiting times in two (left diagram) and three (right diagram) timesteps after the current eruption. For two timesteps (and for even timesteps in general), note that short duration eruptions don't coincide with short waiting times, whereas for three timesteps (and for odd timesteps in general), the orange points are much more similar to the blue points. This shows again that there is some conditional dependence between the two variables, and suggesting there is some underlying states that the geyser is moving between, which are conditionally dependent on states more than one timestep in the past.




Based on these observations we can consider the following distributions to model these data:

1. Multivariate Normal - Based on Observation/plot 1, not suitable - the plot of all data points clearly shows at least two clustered areas, suggesting that a model with a single mean is not accurate.

2. Mixture of Gaussians - based on Observation 1, the time dependence of the data suggests the modelling assumptions of i.i.d data of an MoG model is not suitable.

3. Markov Chain - Based on the observation that the geyser seems to be moving between discrete and separate states, a Markov chain might seem to be a reasonable model. However, as shown by Observation 4, there seem to be longer range dependencies (more than one time step between the variables) indicating that the outputs are not conditionally independent of the outputs from more than one time-steps previously. A Markov chain cannot capture this long range dependence of the outputs and is therefore not suitable.

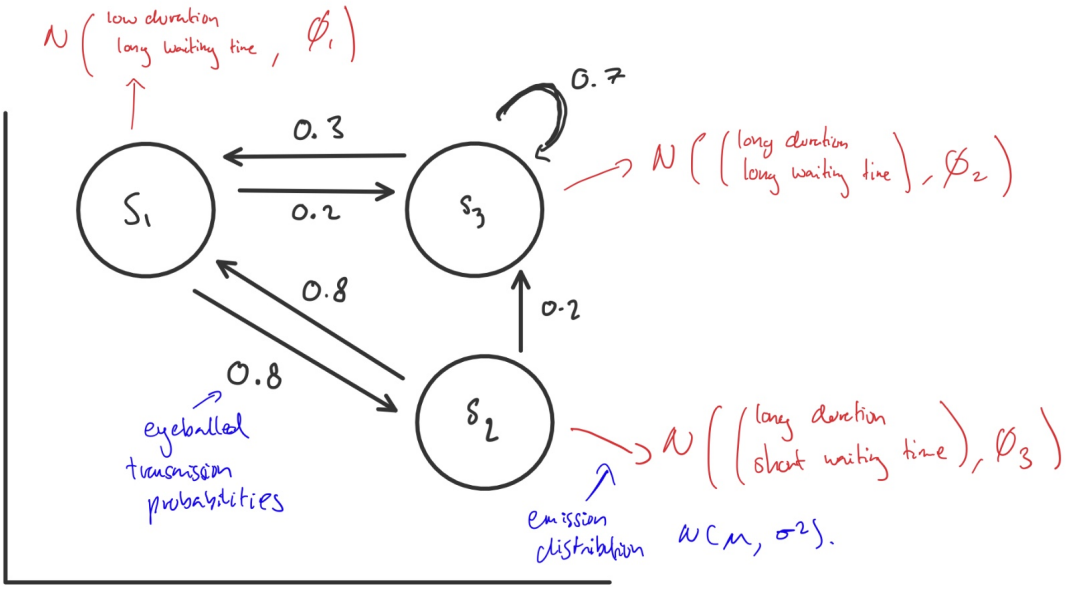
4. HMM - An HMM seems to be a suitable model. It satisfies the long range dependencies of the different outputs (Observation 4), and the emission probabilities of waiting

time and duration can be modelled by mixtures of Gaussians (Observation 2) whose parameters are dependent on the current state, i.e by selecting the weights of the mixtures.

5. LGSSM - The criteria for an LGSSM is that the hidden states are multivariate Gaussian distributions. However based on the plot of points over time (video) (Observation 1), the underlying output emissions tend to take discrete jumps between the different states, indicating that using a continuous Gaussian to model the transitions between states is the appropriate.

6. Stochastic Linear Dynamical System: A SLDS does not seem to applicable, since the hidden states do not seem to be related to the outputs by first order differential equation and instead by Gaussian mixtures. (Observation 2).

Based on this analysis, we conclude that a HMM is the most appropriate model. It seems that 3 discrete states accurately model the data: we can think of the HMM as a MoG model with the mixtures corresponding to 3 different states across time. The outputs seem to be Gaussian (Observation 2), and the choice of 3 states is for the following reasons based on Observation 1: short eruption eruptions are preceded by long waiting times exclusively (one mixture), but long duration eruptions are preceded by both short an long waiting times, with discrete clusters (two mixtures). A suggested HMM is given below:




Part b)
Based on the following observations we construct a HMM.

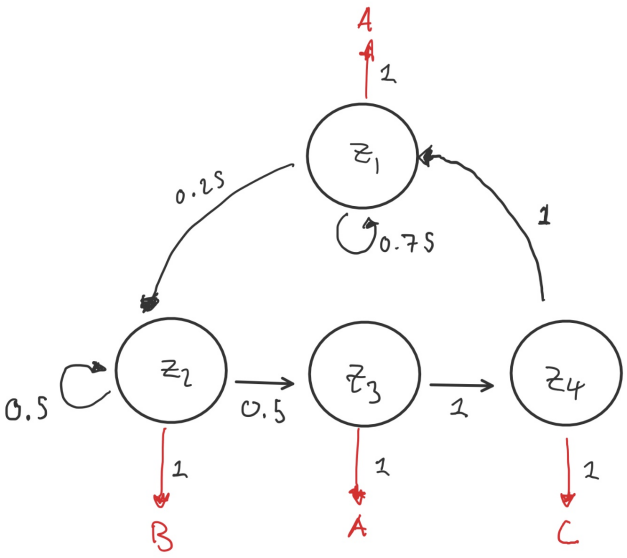
1. The sequence BACA seems to be 'deterministic' i.e. after encountering an A after a B, the sequence CA follows every time (around 25 times with 0 counter examples)

2. There are long sequences of As followed by shorter sequences of Bs (terminated by an A)

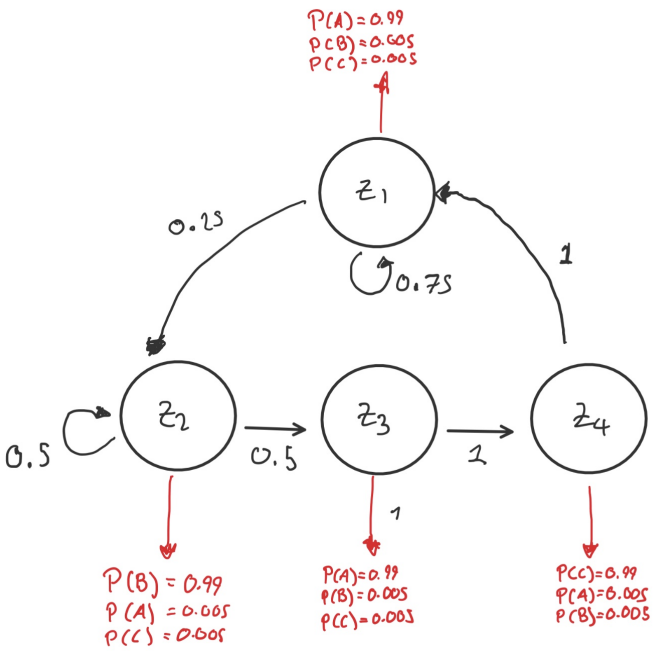
3. No Bs follows Cs and vice versa.

4. To give rough estimates for the probabilities, for As not between a B and a C, an A is followed by another A roughly 75% of the time. Similarly, a B is followed by a B roughly 50% of the time.

Hence, we provide a HMM with emission probabilities of 1, transmission probabilities of 1 between states BACA, and a state emitting A with a high self-returning transmission probability. The transmission probabilities have been computed by counting the lengths of sequences (for instance output A to A occurs frequently so the state emitting just an A is given a high probability (0.75) to stay in that state). Since we have 4 states, the initial state probabilities are given the same value, 0.25, since we are equally as likely to start at the beginning of any sequence.




However, this model essential reduces to a regular Markov Model and provides no scope for the model to account for new data to violate these rules and therefore does not generalise well. There are three option from here: change the number of states, change the transmission probabilities or/and change the emission probabilities. The simplest solution that seems to capture most of the original structure of the data was to add small emission probabilities to the previously excluded letters for each state. The model could then account for new unexpected data by these small emission probabilities, and an algorithm could potentially update these probabilities as a result. The resulting HMM is shown here:




**Problem 3.**

**Solution** Part a) Note that in order for $q(s)$ to put its mass into one setting, restrict $q(s)$ to the class of delta functions, so that the free energy becomes:

$$\begin{aligned}\mathcal{F}(q,\theta) &= \sum_{S_{1:T}} q(s) log(p(s, x_{1:T}|\theta)) - \sum_{s_{1:T}} q(s) log(q(s)) \\ &= \sum_{S_{1:T}} \delta_{s_{1:T}, s'_{1:T}} log(p(s_{1:T}, x_{1:T}|\theta)) - \sum_{s_{1:T}} \delta_{s_{1:T}, s'_{1:T}} log(\delta_{s_{1:T}, s'_{1:T}}) \\ &= \sum_{S_{1:T}} \delta_{s_{1:T}, s'_{1:T}} log(p(s_{1:T}, x_{1:T}|\theta)) + H(q)\end{aligned}$$

note that $H(q)$ is well defined since for $s_{1:T} \neq s'_{1:T}$, then $\delta_{s_{1:T}, s'_{1:T}} = 0$ and for $s_{1:T} = s'_{1:T}$, then $log(\delta_{s_{1:T}, s'_{1:T}}) = 0$, i.e. $H(q) = 0$.

$$\therefore \mathcal{F}(q,\theta) = \sum_{S_{1:T}} \delta_{s_{1:T}, s'_{1:T}} log(p(s_{1:T}, x_{1:T}|\theta)) = log(p(s'_{1:T}, x_{1:T}|\theta))$$

i.e. the Free Energy is just the log-joint (rather than the expected log joint). Therefore to optimise the Free Energy, just maximise the log joint by choosing a maximising $s'_{1:T}$. But by Bayes Rule:

$$log(p(s'_{1:T}, x_{1:T}|\theta)) = log(p(s'_{1:T}|x_{1:T}, \theta) + log(p(x_{1:T}|\theta))$$

$$\implies argmax_{s'_{1:T}} log(p(s'_{1:T}, x_{1:T}|\theta)) = argmax_{s'_{1:T}} log(p(s'_{1:T}|x_{1:T}, \theta))$$
$$:= s^*_{1:T} \text{ (by definition, and log is monotonic increasing)}$$

i.e the choice of $s^*_{1:T}$ maximises the free energy. $\therefore$ since the joint $p(s^*_{1:T}, x_{1:T}|\theta)$ is a probability distribution, it is bounded by 1 and hence if we show the free energy is increasing then the algorithm is guaranteed to converge. (since increasing + bounded $\implies$ convergence) By the Maximisation Step, differentiate w.r.t $\theta$ to find maximising $\theta^{new}$:

$$\mathcal{F}(q, \theta^{old}) = log(p(s^*_{1:T}, x_{1:T}|\theta^{old})) \leq log(p(s^*_{1:T}|x_{1:T}, \theta^{new}) = \mathcal{F}(q, \theta^{new})$$

Part b) It will not converge to a maximum of the likelihood. Instead, since the free energy is given by just the log-joint, it will converge to a (local) maximum of the log-joint. Relating this to Viterbi, the analogue is that the algorithm will converge to the most likely sequence of states, rather than in Baum-Welch the sequence of most likely states.

**Problem 4.**

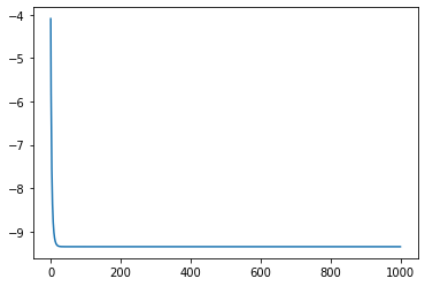
**Solution** Each colored line shows the posterior mean estimate of one dimension of each state (the states are 4 dimensional Gaussians) for all 1000 states.


(a) Filtering (b) Smoothing

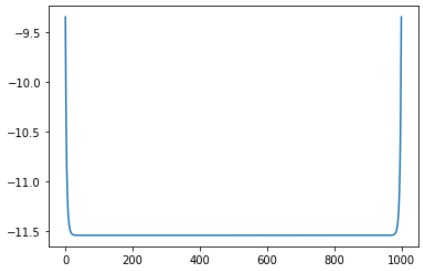
Figure 4: Posterior mean estimates by Kalman filtering vs smoothing

The filtering lines are the result of one forward pass of the algorithm, which recursively calculates the (marginal) posterior estimates. They are 'spiky' because each state estimate only depends on the state transition and observation densities, and the posterior density from the previous time instance. The smoothing lines are the result of one forward pass as well as one backward pass of the algorithm, and hence are smooth because the entire output data $x_{1:T}$ are now presumed known before computation.

The log det of the variances can be seen as a measure of 'differential entropy' for a multivariate Gaussian and therefore the plots show the rate of 'average surprisal' of each state's distribution. The salient point in Figure 5 (a) is that the initial state $y_1$'s differential entropy is much lower than the the subsequent states' $y_{2:T}$. Effectively this means that there is much less uncertainty about the initial state compared to the subsequent states after one forward sweep of the Expectation algorithm. Comparatively, for Kalman Smoothing Figure




(a) Filtering




(b) Smoothing

Figure 5: Log det of posterior variance estimates by Kalman filtering vs smoothing

5 (b) there is both a forward and backward recursion step, and so there is less uncertainty around the initial state $y_1$ and final state $y_T$. This makes sense intuitively because the state posterior estimates are recursively computed from the initial and final states, and hence we become less certain about the estimates for the in-between states.

Part b) Note that $p(x_t|z_t) \propto |R|^{-\frac{1}{2}} exp(-\frac{1}{2}(x_t - Cz_t)^T R^{-1}(x_t - Cz_t)$

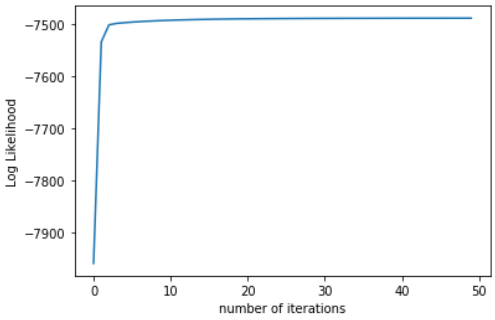
$$
\begin{aligned}
\implies R_{new} &= \underset{R}{\text{argmax}} < \sum_t logp(x_t|z_t) >_q \\
&= \underset{R}{\text{argmax}} \langle \sum_t \{-\frac{1}{2} ln|R| - \frac{1}{2}(x_t - Cz_t)^T R^{-1}(x_t - Cz_t)\} \rangle_q \\
&= \underset{R}{\text{argmax}} \{-\frac{T}{2} ln|R| - \frac{1}{2} \sum_t (x_t^T R^{-1} x_t - 2x_t^T R^{-1} C < z_t > + < z_t^T C^T R^{-1} z_T >)\} \\
&= \underset{R}{\text{argmax}} \{\frac{T}{2} ln|R^{-1}| - \frac{1}{2} \sum_t Tr(x_t^T R^{-1} x_t) + Tr(C \sum_t < z_t > x_t^T R^{-1}) - \\
&- \frac{1}{2}(C^T R^{-1} C \sum_t < z_t z_t^T >)\} \\
&\text{applying } \frac{\partial}{\partial R^{-1}} \\
&\underset{\text{Matrix Cookbook}}{=} \frac{T}{2} R_{new} - \frac{1}{2} \sum_t x_t x_t^T + (C \sum_t < z_t > x_t^T)^T - \frac{1}{2}(C \sum_t < z_t z_t^T >^T C^T)^T \\
&= \frac{T}{2} R_{new} - \frac{1}{2} \sum_t x_t x_t^T + \sum_t x_t < z_t >^T C^T - \frac{1}{2} \underbrace{C \sum_t < z_t z_t^T >}_{= \sum_t x_t < z_t >^T} C^T \\
&= \frac{T}{2} R_{new} - \frac{1}{2} \sum_t x_t x_t^T + \sum_t x_t < z_t >^T C_{new}^T - \frac{1}{2} \sum_t x_t < z_t >^T C_{new}^T \overset{!}{=} 0
\end{aligned}
$$

$$\implies R_{new} = \frac{1}{T}\{\sum_{t=1}^{T} x_t x_t^T - \sum_{t=1}^{T}(x_t < z_t^T >)C_{new}^T\}$$

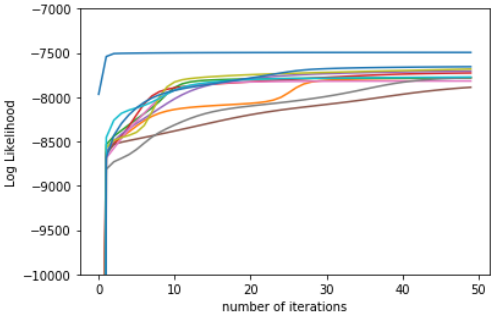
Now note that $p(z_{t+1}|z_t) \propto |Q|^{-\frac{1}{2}} exp(-\frac{1}{2}(z_{t+1} - Az_t)^T Q^{-1}(z_{t+1} - Az_t)$. Hence (skipping a few steps since the derivation is very similar):

$$\begin{aligned}
\implies Q_{new} &= \underset{Q}{\text{argmax}} < \sum_{t=1}^{T-1} logp(z_{t+1}|z_t) >_q \\
&= \underset{Q}{\text{argmax}}(\frac{1}{2} log|Q^{-1}|(T-1) - \frac{1}{2}\sum_t Tr(Q^{-1} < z_{t+1} z_{t+1}^T >) + Tr(A\sum_t < z_t z_{t+1}^T > Q^{-1}) \\
&\quad - \frac{1}{2} Tr(A^T Q^{-1} A \sum_t < z_t z_t^T >) \\
\frac{\partial}{\partial Q^{-1}} &= \frac{T-1}{2} Q - \frac{1}{2}\sum_t < z_{t+1} z_{t+1}^T > + (A\sum_t < z_t z_{t+1}^T >)^T - \frac{1}{2}(A\sum_t < z_t z_{t+1}^T > A^T)^T \\
&= \frac{T-1}{2} Q - \frac{1}{2}\sum_{t=2}^{T} < z_t z_t^T > + \sum_{t=1}^{T-1} < z_{t+1} z_t^T > A_{new}^T - \frac{1}{2}\sum_{t=1}^{T-1} < z_{t+1} z_t^T > A_{new}^T = 0 \\
\implies Q_{new} &= \frac{1}{T-1}\{\sum_{t=2}^{T} < z_t z_t^T > - \sum_{t=2}^{T} < z_t z_{t-1} > A_{new}^T\}
\end{aligned}$$

See code attached. The required plots are given below:




(a) Log Likelihood using given A,Q, C, R.

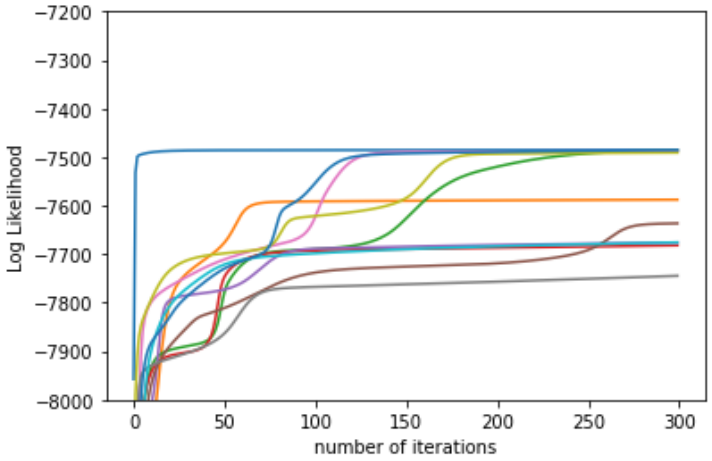



(b) Log Likelihood using given and 10 random A, Q, C, R (zoomed in)

Figure 6: Plots of log likelihood of data after 50 iterations of EM

The log-likelihood using the given generating parameters does not converge immediately because the initial state $y_1$ is still initialised at random, whereas the generated data $x_{1:T}$ have been generated with different (random) seeds $y_1$, and therefore the algorithm still needs to learn the optimal densities for $y_1$
The interesting thing about the plots - Figure 6 (b) - is that:
Firstly, the log likelihoods of the random initialisation are bounded by the log likelihood of the given initialisation. This is true because clearly, the EM algorithm learns the exact parameters for the given initialisations, so for the random starting points it is not possible to do better than that. Secondly, for the random starting points the algorithm 'converges' (not quite converges since the log-likelihood keeps increasing just at a slower rate) to different log-likelihood values. This can be explained by the fact that LGSSM likelihood functions have multiple local maxima, and so different starting points will lead to the algorithm finding different local maxima.

For 300 iterations we get:




This plot shows that from different random starting conditions, the log-likelihood can get 'trapped' for quite some time before beginning to increase again, and that it is still unclear whether one random iteration will reach a local maximum or continue to increase. Therefore the number of iterations required to a local maximum can be rather large.