Probabilistic & Unsupervised Learning: Summative Assignment

Enric Balaguer Rodon

November 15, 2023

Contents

1 Exercise 1

My code for this exercise is in the submitted .zip file.

1.a

A multivariate Gaussian model would not be appropriate for this set of images because it is continuous, not discrete.
In this example where our data can have two discrete outcomes; black or white, 0 or 1, a discrete distribution would suit our data better. Specifically a Bernoulli distribution.

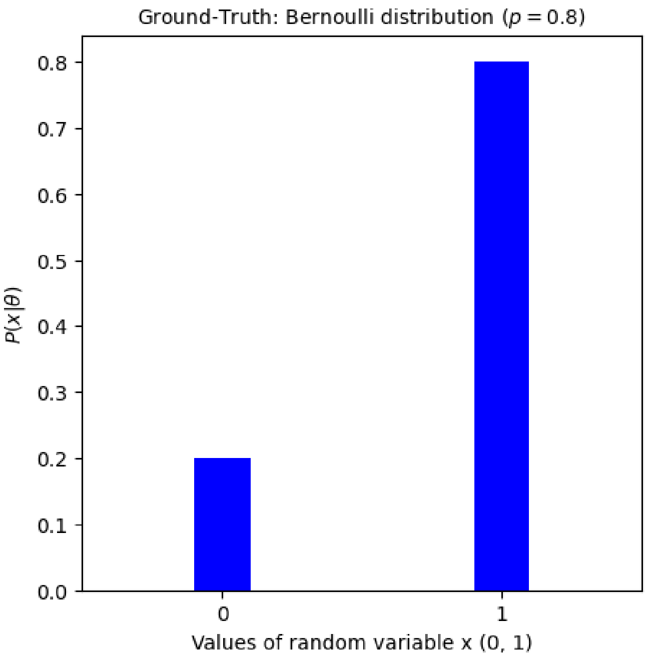


Figure 1: Example where each pixel has 20% chance of being black (0) and 80% chance of being white (1)

A multivariate Gaussian would center itself somewhere in between the two possible outcomes, with large enough tails to attempt to satisfy both outcomes, but failing to do so.

1.b

In order to acquire ML estimation of $\mathbf{p}$, we must:

$$\hat{\mathbf{p}}^{ML} = \arg\max_{P} P(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N|\mathbf{p}) \tag{1}$$

Thus, we start by setting the joint probability for all $\vec{x}$:

$$P(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N|\mathbf{p}) = \prod_{n=1}^{N}\prod_{d=1}^{D} p_d^{x_d^{(n)}} (1 - p_d)^{(1-x_d^{(n)})} \tag{2}$$

We log the joint probability:

$$\log(P(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N|\mathbf{p})) = \sum_{n=1}^{N}\sum_{d=1}^{D} x_d^{(n)} \log(p_d) + (1 - x_d^{(n)}) \log(1 - p_d) \tag{3}$$

Proceed by finding the values of **p** that maximise the log joint likelihood:

$$\boldsymbol{\nabla}_{\mathbf{p}} \log P(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N|\mathbf{p}) = 0 \tag{4}$$

Please note in equation 5 below, all terms in the sum will be 0 except for when $d = x$.

$$\frac{\partial}{\partial p_x} \sum_{d=1}^{D} x_d^{(n)} \log(p_d) + (1 - x_d^{(n)}) \log(1 - p_d) = \frac{x_x^{(n)}}{p_x} - \frac{1 - x_x^{(n)}}{1 - p_x} \tag{5}$$

That allows us to re-write the expression in equation 4 as:

$$\sum_{n=1}^{N}\sum_{d=1}^{D} \frac{x_d^{(n)}}{p_d} - \frac{1 - x_d^{(n)}}{1 - p_d}\hat{p}_d = 0 \tag{6}$$

Thus, for each entry of **p** we have:

$$\sum_{n=1}^{N} \frac{x_d^{(n)}}{p_d} - \frac{1 - x_d^{(n)}}{1 - p_d} = 0 \tag{7}$$

We remember that $x_d^{(n)}$ can only be $\{0, 1\}$, allowing us to sum over n directly:

$$\frac{N_d}{p_d} - \frac{1 - x_d^{(n)}}{1 - p_d} = 0 \tag{8}$$

Where N is the number of images and N_d is the sum over every image's $d - th$ pixel.
From here, we can solve for p_d:

$$\frac{N_d}{p_d} - \frac{1 - x_d^{(n)}}{1 - p_d} = 0 \Rightarrow \frac{N_d(1 - p_d)}{p_d(1 - p_d)} - \frac{(1 - x_d^{(n)})p_d}{(1 - p_d)p_d} = 0 \Rightarrow N_d - \cancel{N_d p_d} = p_d N - \cancel{N_d p_d} \tag{9}$$

Finally acquiring:

$$p_d = \frac{N_d}{N} = \frac{\sum_{n=1}^{N} x_d^{(n)}}{N} \tag{10}$$

Where p_d is each entry of the **p** vector: $\mathbf{p} = (p_1, p_2, ..., p_D)$

1.c

In order to acquire MAP estimation of **p**, we must:

$$\hat{\mathbf{p}}^{MAP} = \arg\max_P P(\mathbf{p}|\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N) = \arg\max_P \frac{P(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N|\mathbf{p})P(\mathbf{p})}{P(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N)}) \tag{11}$$

Where we used Bayes' Theorem to re-write it to the RHS form.
Here we can recognise the denominator has no **p** dependence, allowing us to deal with the nominator only.

$$P(\mathbf{x}_1, ..., \mathbf{x}_N|\mathbf{p})P(\mathbf{p}) = \prod_{n=1}^{N}\prod_{d=1}^{D} p_d^{x_d^{(n)}}(1-p_d)^{(1-x_d^{(n)})}\prod_{d=1}^{D}\frac{1}{B(\alpha,\beta)}p_d^{(\alpha-1)}(1-p_d)^{(\beta-1)} \tag{12}$$

After "extracting" $\frac{1}{B(\alpha,\beta)}$ out of the three products and logging the entire expression, we acquire $\log(P(\mathbf{x}_1, ..., \mathbf{x}_N|\mathbf{p})P(\mathbf{p}))$ to be:

$$\log\left(\frac{1}{B^{ND^2}}\right)+\sum_{n=1}^{N}\sum_{d=1}^{D}x_d^{(n)}\log(p_d)+(1-x_d^{(n)})\log(1-p_d)+\sum_{d=1}^{D}(\alpha-1)\log(p_d)+(\beta-1)\log(1-p_d) \tag{13}$$

We proceed by finding the values of $\mathbf{p}$ that maximise the log MAP likelihood:

$$\boldsymbol{\nabla}_{\mathbf{p}}\log(P(\mathbf{x}_1, ..., \mathbf{x}_N|\mathbf{p})P(\mathbf{p})) = 0 \tag{14}$$

Noting once again what is mentioned in 5, we can re-write equation 14 to be:

$$\sum_{n=1}^{N}\sum_{d=1}^{D}\left(\frac{x_d^{(n)}}{p_d}-\frac{1-x_d^{(n)}}{1-p_d}\right)\hat{p}_d+\sum_{d=1}^{D}\left(\frac{\alpha-1}{p_d}-\frac{\beta-1}{1-p_d}\right)\hat{p}_d = 0 \tag{15}$$

Thus, for each entry of $\mathbf{p} = (p_1, p_2, ..., p_D)$ we have:

$$\sum_{n=1}^{N}\left(\frac{x_d^{(n)}}{p_d}-\frac{1-x_d^{(n)}}{1-p_d}\right)+\left(\frac{\alpha-1}{p_d}-\frac{\beta-1}{1-p_d}\right)= 0 \tag{16}$$

We proceed with algebra in order to acquire an expression for p_d. Once again, we use $N_d = \sum_{n=1}^{N} x_d^{(n)}$ for simplicity.

$$0 = \frac{N_d}{p_d}-\frac{1-x_d^{(n)}}{1-p_d}+\frac{\alpha+1}{p_d}-\frac{\beta-1}{1-p_d} \tag{17}$$

$$0 = \frac{N_d+(\alpha-1)(1-p_d)}{\cancel{p_d(1-p_d)}}-\frac{p_d((N-N_d)+(\beta-1))}{\cancel{p_d(1-p_d)}} \tag{18}$$

$$0 = N_d+\alpha-1-\cancel{N_d p_d}-\alpha p_d+p_d-Np_d+\cancel{N_d p_d}-\beta p_d+p_d \tag{19}$$

$$0 = N_d+\alpha-1+p_d(-N-\beta+1-\alpha+1) \tag{20}$$

Finally acquiring the desired result:

$$p_d = \frac{N_d+\alpha-1}{N+\beta+\alpha-2} = \frac{\sum_{n=1}^{N}x_d^{(n)}+\alpha-1}{N+\beta+\alpha-2} \tag{21}$$

1.d

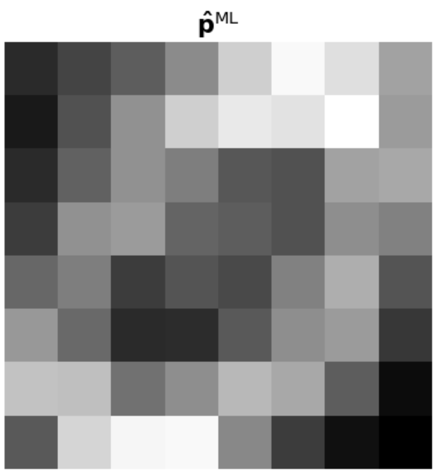


Figure 2: Maximum likelihood parameters of a multivariate Bernoulli from binarydigits.txt

1.e

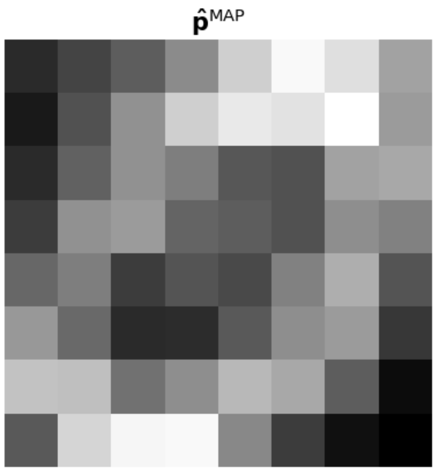


Figure 3: MAP parameters of a multivariate Bernoulli from binarydigits.txt

Evidently, maximum likelihood returns a best estimate based solely on the likelihood. In the other hand, MAP allows us to input prior information to come up with a different best estimate. Thus, if no prior information at all on the model is available, ML must be used. However, if prior information is available a MAP estimation might yield better results, the problem is determining if such prior information will add to the validity of such estimate or subtract from it.
I would say MAP estimation can yield better results with a prior that has been tested through another algorithm or model, or through a prior that is the conjugate prior of the likelihood. If there is no prior information about the model I would say ML is a safer choice rather than estimating MAP with perhaps a wrong/bias prior.

2 Exercise 2

My code for this exercise is in the submitted .zip file.
We want to calculate the posterior for every model:

$$P(\mathbf{p} \mid \mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)}) = \frac{P(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)} \mid \mathbf{p})P(\mathbf{p})}{P(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)})} \tag{22}$$

The denominator in equation 22 is the same for every model, as it is the sum of the three model's nominator. Thus, let's first acquire an expression for each model's nominator. The prior is given to us in the exercise, $P(\mathbf{p}) = \frac{1}{3}$, no matter the model. We now focus on the likelihoods.
Likelihood for model a) can be acquired analytically:
Reminder: $\mathbf{p}^a = (\frac{1}{2}, ..., \frac{1}{2}) \in \mathbb{R}^D$, where I use $p^a = \frac{1}{2}$

$$\begin{aligned} P(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)} \mid \mathbf{p}^a) &= \prod_{n=1}^{N}\prod_{d=1}^{D} (p^a)^{x_d^{(n)}}(1-p^a)^{(1-x_d^{(n)})} & (23)\\ &= \prod_{n=1}^{N}\prod_{d=1}^{D} \left(\frac{1}{2}\right)^{x_d^{(n)}} \left(\frac{1}{2}\right)^{1-x_d^{(n)}} & (24)\\ &= \left(\frac{1}{2}\right)^{ND} & (25)\end{aligned}$$

Likelihood for model b) can be acquired following the same procedure:
Reminder: $\mathbf{p}^b = (p^b, ..., p^b) \in \mathbb{R}^D$, where $p^b \sim U(0, 1)$

$$\begin{aligned} P(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)} \mid \mathbf{p}^b) &= \int_0^1 \prod_{n=1}^{N}\prod_{d=1}^{D} (p^b)^{x_d^{(n)}}(1-p^b)^{(1-x_d^{(n)})} dp^b & (26)\\ &= \int_0^1 (p^b)^{\sum_{n=1}^N \sum_{d=1}^D x_d^{(n)}}(1-p^b)^{\sum_{n=1}^N \sum_{d=1}^D (1-x_d^{(n)})} dp^b & (27)\\ &= B\left(\alpha_b = \left(1 + \sum_{n=1}^{N}\sum_{d=1}^{D} x_d^{(n)}\right), \beta_b = \left(1 + \sum_{n=1}^{N}\sum_{d=1}^{D}(1 - x_d^{(n)}\right)\right) & (28)\end{aligned}$$

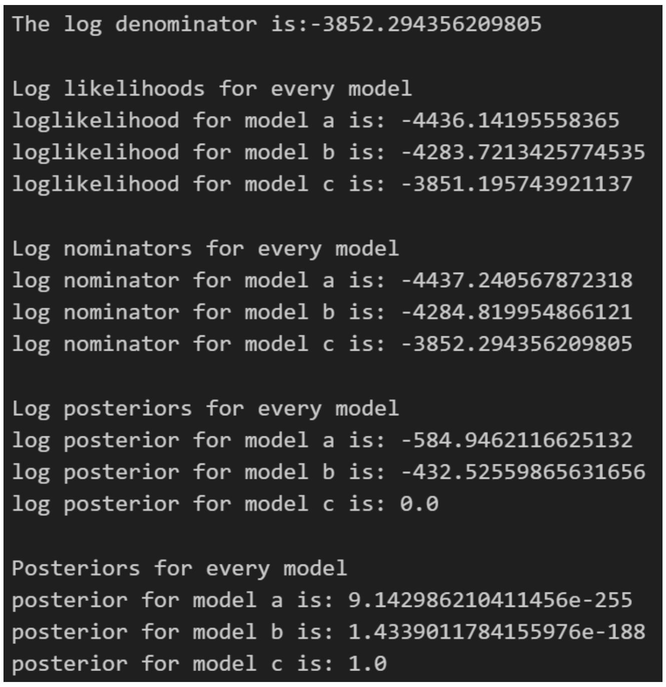
Explanation: Since p^b is uniformly distributed, we must integrate the likelihood function over the limits of the uniform distribution, yielding 26. We recognise p^b has no n nor d dependence, allowing us to apply the products directly, as shown in equation 27. Finally, we recognise our expression resembles the beta function in integral form, allowing us to re-write it as such in equation 28.
We proceed by acquiring likelihood for model c).
Reminder: $\mathbf{p}^c = (p_1^c, ..., p_D^c) \in \mathbb{R}^D$, where $p_d^c \sim U(0, 1)$.

$$\begin{aligned} P(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)} \mid \mathbf{p}^c) &= \prod_{d=1}^{D}\int_0^1 \prod_{n=1}^{N} (p_d^c)^{x_d^{(n)}}(1-p_d^c)^{(1-x_d^{(n)})} dp_d^c & (29)\\ &= \prod_{d=1}^{D}\int_0^1 (P_d^c)^{\sum_{n=1}^N x_d^{(n)}}(1-p_d^c)^{\sum_{n=1}^N (1-x_d^{(n)})} & (30)\\ &= \prod_{d=1}^{D} B\left(\alpha_c = \left(1 + \sum_{n=1}^{N} x_d^{(n)}\right), \beta_c = \left(1 + \sum_{n=1}^{N}(1 - x_d^{(n)})\right)\right) & (31)\end{aligned}$$

Explanation: In this model we have D uniformly sampled variables(p_d^c), thus we need to integrate the likelyhood expression D times over their respective uniform distribution limits. I use the commutative property of multiplication to formulate the equation shown in 29. We then recognise p_d^c has no n dependence and apply the products directly to yield equation 30. Once again, we recognise our expression resembles the beta function in integral form, allowing us to re-write it as such in equation 31.
Now that we have an expression for the likelihood of each model, we can implement them through in Python and compute the posterior for every model.
I run into numerical issues. We use logarithms and logsumexp trick to help us. Here are the log likelihoods implemented in my code for every model:

- $\log\big(P(\mathbf{x}^{(1)},...,\mathbf{x}^{(N)} \mid \mathbf{p}^{a})\big) = ND\log\big(\frac{1}{2}\big)$
- $\log\big(P(\mathbf{x}^{(1)},...,\mathbf{x}^{(N)} \mid \mathbf{p}^{b})\big) = \log(\Gamma(\alpha_b)) + \log(\Gamma(\beta_b)) - \log(\Gamma(\alpha_b+\beta_b))$
  Where α_b and β_b are defined in equation 28
- $\log\big(P(\mathbf{x}^{(1)},...,\mathbf{x}^{(N)} \mid \mathbf{p}^{b})\big) = \sum_{d=1}^{D}\log(\Gamma(\alpha_c)) + \log(\Gamma(\beta_c)) - \log(\Gamma(\alpha_c+\beta_c))$
  Where α_c and β_c are defined in equation 31

Here are all the results I acquire:



```
The log denominator is:-3852.294356209805

Log likelihoods for every model
loglikelihood for model a is: -4436.14195558365
loglikelihood for model b is: -4283.7213425774535
loglikelihood for model c is: -3851.195743921137

Log nominators for every model
log nominator for model a is: -4437.240567872318
log nominator for model b is: -4284.819954866121
log nominator for model c is: -3852.294356209805

Log posteriors for every model
log posterior for model a is: -584.9462116625132
log posterior for model b is: -432.52559865631656
log posterior for model c is: 0.0

Posteriors for every model
posterior for model a is: 9.142986210411456e-255
posterior for model b is: 1.4339011784155976e-188
posterior for model c is: 1.0
```

Figure 4: Summary of results acquired through Python

Unfortunately, I am unaware of a way to fix the numerical issue I am encountering when calculating the denominator. In the attached code I use scipy's logsumexp function to show that even when scaling logsumexp, the numerical error still persists. I do not use scipy's logsumexp function anywhere else in my code, I use it here to show that even if I were to implement scaling, I would still face the same numerical issue.
Given such issue, I resort to providing the relative probabilities as my final answer. Since the denominator is constant for all three models, I provide the log posterior of every model up to a

constant:

Model a)

$$\log\Big(P(\mathbf{p}^a \mid \mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)})\Big) \propto \log\Big(P(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)} \mid \mathbf{p}^a)\Big) + \log(P(\mathbf{p}^a)) = -4437.240567872318$$

Model b)

$$\log\Big(P(\mathbf{p}^b \mid \mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)})\Big) \propto \log\Big(P(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)} \mid \mathbf{p}^b)\Big) + \log\Big(P(\mathbf{p}^b)\Big) = -4284.819954866121$$

Model c)

$$\log\Big(P(\mathbf{p}^c \mid \mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)})\Big) \propto \log\Big(P(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)} \mid \mathbf{p}^c)\Big) + \log(P(\mathbf{p}^c)) = -3852.294356209805$$

3 Exercise 3

My code for this exercise is in the submitted .zip file.

3.a

Taking into account that the likelihood for any given image is the following:

$$P(\mathbf{x}^{(n)} \mid \boldsymbol{\pi}, \mathbf{P}) = \sum_{k=1}^{K} P(\mathbf{x}^{(n)} \mid s^{(n)} = k) P(s^{(n)} = k) = \sum_{k=1}^{K} P_k(\mathbf{x}^{(n)}; \mathbf{p}_k)\pi_k \tag{32}$$

Where $\mathbf{p}_k$ is the $k-th$ row of matrix $\mathbf{P}$ and P_k is the $k-th$ multivariate Bernoulli distribution. Then we can use 32 to acquire the likelihood for all N images:

$$P(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)} \mid \boldsymbol{\pi}, \mathbf{P}) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \prod_{d=1}^{D} (p_{kd})^{\mathbf{x}_d^{(n)}} (1 - p_{kd})^{(1-\mathbf{x}_d^{(n)})} \tag{33}$$

3.b

Using the same notation as the most RHS equation in 32:

$$\begin{align} r_{nk} = P(s^{(n)} = k \mid \mathbf{x}^{(n)}, \boldsymbol{\pi}, \mathbf{P}) &= \frac{P_k(\mathbf{x}^{(n)}; \mathbf{p}_k)\pi_k}{\sum_{k=1}^{K} P_k(\mathbf{x}^{(n)}; \mathbf{p}_k)\pi_k} \tag{34} \\ &= \frac{\pi_k \prod_{d=1}^{D} (p_{kd})^{x_d^{(n)}} (1 - p_{kd})^{(1-x_d^{(n)})}}{\sum_{k=1}^{K} \pi_k \prod_{d=1}^{D} (p_{kd})^{x_d^{(n)}} (1 - p_{kd})^{(1-x_d^{(n)})}} \tag{35} \end{align}$$

3.c

I start by re-writing the expression given in the exercise to be:

$$\arg\max_{\boldsymbol{\pi}, \mathbf{P}} \left\langle \sum_{n=1}^{N} \log\Big(P(\mathbf{x}^{(n)}, s^{(n)} \mid \boldsymbol{\pi}, \mathbf{P})\Big) \right\rangle_{q(s^{(n)})} = \arg\max_{\boldsymbol{\pi}, \mathbf{P}} \left\langle \log\Big(P(\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)}, s^{(1)}, ..., s^{(N)} \mid \boldsymbol{\pi}, \mathbf{P})\Big) \right\rangle_{q(s^{(n)})} \tag{36}$$

We can now use the definition given to us in Slide 125/287 of the notes. 'In the M-step we optimize the sum (since $s^{(n)}$ is discrete)'. Thus yielding the following E-step to optimise:

$$\text{E-step} = \sum_{n=1}^{N} \sum_{k=1}^{K} q(s^{(n)} = k) \log\Big(P(s^{(n)}) P(\mathbf{x}^{(n)} \mid s^{(n)} = k)\Big) \tag{37}$$

We proceed to re-write such expression:

$$\text{E-step} = \sum_{n=1}^{N}\sum_{k=1}^{K} q(s^{(n)}=k)\log(\pi_k P_k(\mathbf{x};\mathbf{p}_k)) \tag{38}$$

$$= \sum_{n=1}^{N}\sum_{k=1}^{K} q(s^{(n)}=k)\log\left(\pi_k \prod_{d=1}^{D} (p_{kd})^{x_d^{(n)}}(1-p_{kd})^{(1-x_d^{(n)})}\right) \tag{39}$$

$$= \sum_{n=1}^{N}\sum_{k=1}^{K} q(s^{(n)}=k)\left(\log(\pi_k)+\sum_{d=1}^{D} x_d^{(n)}\log(p_{kd})+(1-x_d^{(n)})\log(1-p_{kd})\right) \tag{40}$$

$$= \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}\left(\log(\pi_k)+\sum_{d=1}^{D} x_d^{(n)}\log(p_{kd})+(1-x_d^{(n)})\log(1-p_{kd})\right) \tag{41}$$

Where, in the last step, we have recognised the fact that $q(s^{(n)}=k)=r_{nk}$, by definition.
We now proceed by finding the optimum by setting the partial derivatives of our E-step to zero (as specified in slide 125 of the notes):

$$\text{Set partial derivatives of E to zero}\begin{cases}\frac{\partial E}{\partial p_{kd}} &= 0\\ \\ \frac{\partial E}{\partial \pi_k} &= 0\end{cases} \tag{42}$$

The solution to $\frac{\partial E}{\partial \pi_k}=0$ is generalised for any mixture of distributions in slide 125 of the notes, thus we can use result acquired there: $\hat{\pi}_k = \frac{1}{N}\sum_{n=1}^{N} r_{nk}$

Note: I should clarify. I am unaware if the result in slide 125 generalises for <u>any</u> mixture of distributions, in other words, I am unaware if there exists a weird mixture of distributions that does not follow such result. <u>However</u>, what is most important is that such result <u>does</u> generalise to our particular case of mixture of multivariate Bernoulli distributions. Given that the partial derivative of our log likelihood over π_k is the same as slide 90 of the notes ($\frac{\partial \mathcal{L}}{\pi_k}=\sum_{n=1}^{N}\frac{r_{nk}}{\pi_k}$) and our constraint is also the same ($\sum_{k=1}^{K}\pi_k=1$), we can generalise the result obtained in slide 125 of the notes.

I don't want to be graded down for this so I proceed to acquire $\frac{\partial E}{\partial \pi_k}=0$ anyways. We need to use Lagrange Multipliers as $\sum_{k=1}^{K}\pi_k=1$ poses a constraint to our solution.

$$\frac{\partial\left(E+\lambda\left(\sum_{k=1}^{K}\pi_k-1\right)\right)}{\partial \pi_k}=0 \tag{43}$$

$$\sum_{n=1}^{N}\frac{r_{nk}}{\hat{\pi}_k}+\lambda=0 \tag{44}$$

We sum the expression over all k to allow us to use our constraint.

$$\sum_{k=1}^{K}\sum_{n=1}^{N} r_{nk}+\sum_{k=1}^{K}\lambda\hat{\pi}_k=0 \tag{45}$$

$$N+\lambda=0 \tag{46}$$

$$\lambda=-N \tag{47}$$

This allows us to substitute found value of λ in equation 44 to yield: $\hat{\pi}_k = \frac{1}{N}\sum_{n=1}^{N} r_{nk}$

We proceed by acquiring $\hat{p}_{kd}$ by solving $\frac{\partial E}{\partial \hat{p}_{kd}=0}$. We first compute the partial derivative. Taking into account π_k is a constant and that, for any function of $f(p_{kd})$:

$$\frac{\partial}{\partial p_{kd}}\left(\sum_{i=1}^{K}\sum_{j=1}^{D} f(p_{kd})\right) = 0 \text{ Except for when } i = k \text{ and } j = 1. \tag{48}$$

We can compute the partial derivative to be:

$$\frac{\partial E}{\partial \pi_k} = \sum_{n=1}^{N} r_{nk}\left(\frac{\mathbf{x}_d^{(n)}}{\hat{p}_{kd}} - \frac{1-\mathbf{x}_d^{(n)}}{1-\hat{p}_{kd}}\right) = 0 \tag{49}$$

From here, we proceed to find an expression for $\hat{p}_{kd}$. We recognise $\hat{p}_{kd}$ has no n dependence:

$$(1-\hat{p}_{kd})\left(\sum_{n=1}^{N} r_{nk}x_d^{(n)}\right) = \hat{p}_{kd}\left(\sum_{n=1}^{N} r_{nk}\left(1-x_d^{(n)}\right)\right) \tag{50}$$

$$\sum_{n=1}^{N} r_{nk}x_d^{(n)} - \hat{p}_{kd}\cancel{\sum_{n=1}^{N} r_{nk}x_d^{(n)}} = \hat{p}_{kd}\sum_{n=1}^{N} r_{nk} - \hat{p}_{kd}\cancel{\sum_{n=1}^{N} r_{nk}x_d^{(n)}} \tag{51}$$

Thus acquiring:

$$\hat{p}_{kd} = \frac{\sum_{n=1}^{N} r_{nk}x_d^{(n)}}{\sum_{n=1}^{N} r_{nk}} \tag{52}$$

3.d

All my code (including responsibilities matrix if desired) is in the submitted .zip file. I tried to implement the many numerical solutions suggested in the exercise but kept failing to do so successfully. After failing a few times, I decided to try and vectorise the EM algorithm to test the waters, and works perfectly fine with just logging the expressions. Thus, if when reading my code you are wondering why I do not use logsumexp or any other numerical solution, I found it to work best without them and just vectorising and logging the expressions (works best because it's more efficient).
Summary of E-Step and M-Step needed to implement.

$$\text{E-Step}\left\{ r_{nk} = P(s^{(n)} = k \mid \mathbf{x}_d^{(n)}, \boldsymbol{\pi}, \mathbf{P}) = \frac{\pi_k \prod_{d=1}^{D}(p_{kd})^{x_d^{(n)}}(1-p_{kd})^{(1-x_d^{(n)})}}{\sum_{k=1}^{K}\pi_k \prod_{d=1}^{D}(p_{kd})^{x_d^{(n)}}(1-p_{kd})^{(1-x_d^{(n)})}} \right. \tag{53}$$

$$\text{M-Step}\begin{cases} \hat{p}_{kd} &= \frac{\sum_{n=1}^{N} r_{nk}x_d^{(n)}}{\sum_{n=1}^{N} r_{nk}} \\ \\ \hat{\pi}_k &= \frac{1}{N}\sum_{n=1}^{N} r_{nk} \end{cases} \tag{54}$$

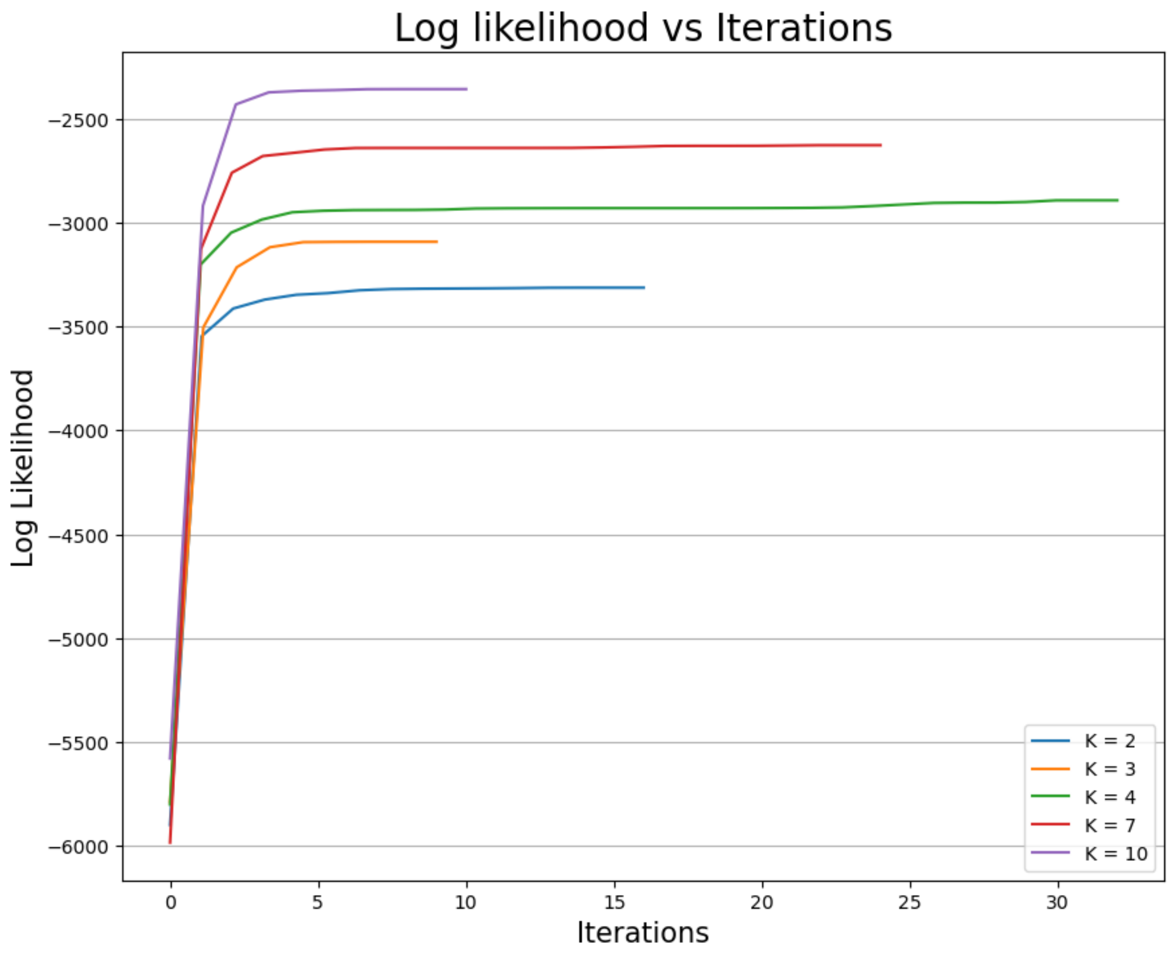


Figure 5: $\pi_{k=2}$ = [0.63008291 0.36991709]
$\pi_{k=3}$ = [0.49018253 0.1599743 0.34984316]
$\pi_{k=4}$ = [0.37026062 0.19997334 0.21001499 0.21975105]
$\pi_{k=7}$ = [0.11 0.17000443 0.10000656 0.17 0.01 0.11999199 0.31999702]
$\pi_{k=10}$ = [0.03999993 0.04 0.13 0.04999913 0.1202727 0.12979295 0.06993446 0.11999989 0.15 0.15000094]

I am unsure how many parameters I need to show in this subsection, π_k is shown above, P_{kd} matrix is shown in below subsection and responsibilities matrix is $N \times K$ and can't display it as an image. Thus, I am supposing we are not meant to show the responsibilities matrix. However, if such responsibilities matrix are desired, they can be found at the end of the report with the code. As explained, I show P_{kd} in the below subsection.

3.e

After running the algorithm a few times starting from randomly chosen initial conditions, I can see the log likelihood for fixed K tends to converge to approximately the same log likelihood value. However, the π_k is quite different each run, although the terms in π_k tend to be similar for low k.

Yes it depends on value K. The log likelihoods shown on figure 5 show good values to which my mixture of multivariate Bernoulli's log likelihood ends up converging to for such fixed value K.

The learned probability vectors as images are shown below:

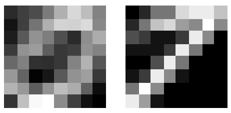


Figure 6: Matrix P when $K = 2$

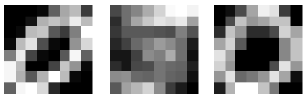


Figure 7: Matrix P when $K = 3$

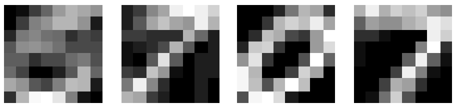


Figure 8: Matrix P when $K = 4$

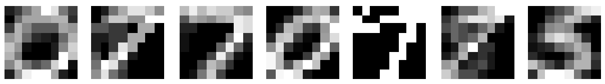


Figure 9: Matrix P when $K = 7$

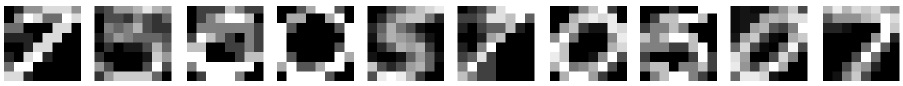


Figure 10: Matrix P when $K = 10$

Please keep in mind, I am unaware if the results attached above stem from an unlucky run. I have been consistently acquiring the majority of the "trained" matrices P vectors to resemble really successfully either 0, 5 or 7.

Finally, I will comment on how well the algorithm works and how might I improve it, as asked.

I would say the EM algorithm I implemented works as desired, when studying the responsibilities matrix, I notice all entries in each row are zero except for one, which is unity, further demonstrating the EM algorithm converges to the desired value requested by the model (assigning one multivariate Bernoulli to recognise each possible digit). However, I find the choice of big K values (as requested in above subsections $K = 7$, $K = 10$) a poor choice for our desired model. Unless we have some previous knowledge that allows us to safely assume big K values, I think such decision would cause our model to over-fit the data. Since our given data-set (appears to) only contain images for the digits 0, 5 and 7, I would assume for any $K < 3$ value to be a horrible decision, and any $K > 3$ value to have to be studied further (unless for really big values of K, then safe to assume a poor choice, if I were obligated to give a range I would say best K is somewhere within $2 < K < 7$, but again, further study is needed).

Why do I suggest having $K > 2$? Given our data-set (appears to) only contain images for the digits 0, 5 and 7, I would assume for the best choice of K to be one that <u>at least</u>, allows us to train three different multivariate Bernoullis, one to recognise each digit. Any other assumption beyond that requires prior knowledge of the system (while always being aware that the higher the value of K, the more risk we are posing our model to overfit the data).

Which brings me to one of the ways I would attempt to improve our model, with prior knowledge of our model. For example, if we knew that half the people write the digit 7 with a line in between and half the people write the digit 7 without such line (as you are seeing it typed here), then it would perhaps be a good idea to increase K by 2 if number 7 is within our dataset (one value of K to develop a multivariate Bernoulli to recognise 7 with the line and another multivariate Bernoulli to recognise 7 without such line).

4 Exercise 4

5 Exercise 5

My code for this exercise is in the submitted .zip file.

5.a

We re-write $P(\mathbf{s})$ as requested by the exercise (in terms of transition matrix):

$$P(\mathbf{s}) = P(s_1)\prod_{i=2}^{n} P(s_i \mid s_{i-1}) = P(s_1)\prod_{\alpha,\beta} \psi(\alpha,\beta)^{N(\alpha,\beta)} \tag{55}$$

Where $\psi(\alpha,\beta) = P(s_i = \alpha \mid s_{i-1} = \beta)$ is the transition probability of going from state β to state α (in our example states = symbols). And $N(\alpha,\beta)$ is the number of times such transition (transition from β to α) happens in our text.
Thus, when attempting to acquire $\psi^{ML}(\alpha,\beta)$, we will be constraint by: $\sum_\beta \psi(\alpha,\beta) = 1$. Because probabilities to go from any β to a single α must sum up to 1.
Thus, we use Lagrange Multipliers to acquire $\psi^{ML}(\alpha,\beta)$:

$$\text{Lagrange Multipliers}\begin{cases}\nabla f + \lambda \nabla g = 0 \\ g = 0\end{cases} \quad \begin{cases}\frac{\partial}{\partial\psi(\alpha,\beta)}\Big(\log(P(\mathbf{s})) + \sum_{i\in s}\lambda_i(1-\sum_\beta \psi(\alpha,\beta))\Big) = 0 \\ \sum_\beta \psi(\alpha,\beta) = 1\end{cases} \tag{56}$$

Using the expression shown in 55, we can express $\log(P(\mathbf{s}))$ to be:

$$\log(P(\mathbf{s})) = \log(P(s_1)) + \sum_{\alpha,\beta} N(\alpha,\beta)\log(\psi(\alpha,\beta)) \tag{57}$$

Allowing us to differentiate $\nabla f + \lambda\nabla g = 0$ directly:

$$\frac{\partial}{\partial\psi(\alpha,\beta)}\left(\log(P(\mathbf{s})) + \sum_{i\in s}\lambda_i(1-\sum_\beta \psi(\alpha,\beta))\right) = \frac{N(\alpha,\beta)}{\psi(\alpha,\beta)} - \lambda = 0 \tag{58}$$

In order to use our constraint to find a solution, we sum over β:

$$\sum_\beta N(\alpha,\beta) = \sum_\beta \lambda\psi(\alpha,\beta) \to \lambda = \sum_\beta N(\alpha,\beta) \tag{59}$$

Using such result for the Lagrange multiplier, we acquire $\psi^{ML}(\alpha,\beta)$ to be:

$$\psi^{ML}(\alpha,\beta) = \frac{N(\alpha,\beta)}{\sum_\beta N(\alpha,\beta)} \tag{60}$$

Note: I think about $\psi(\alpha,\beta)$ and $N(\alpha,\beta)$ as an entry of a 53×53 matrix. We calculate such matrices through the code I attach and acquire the desired table below.

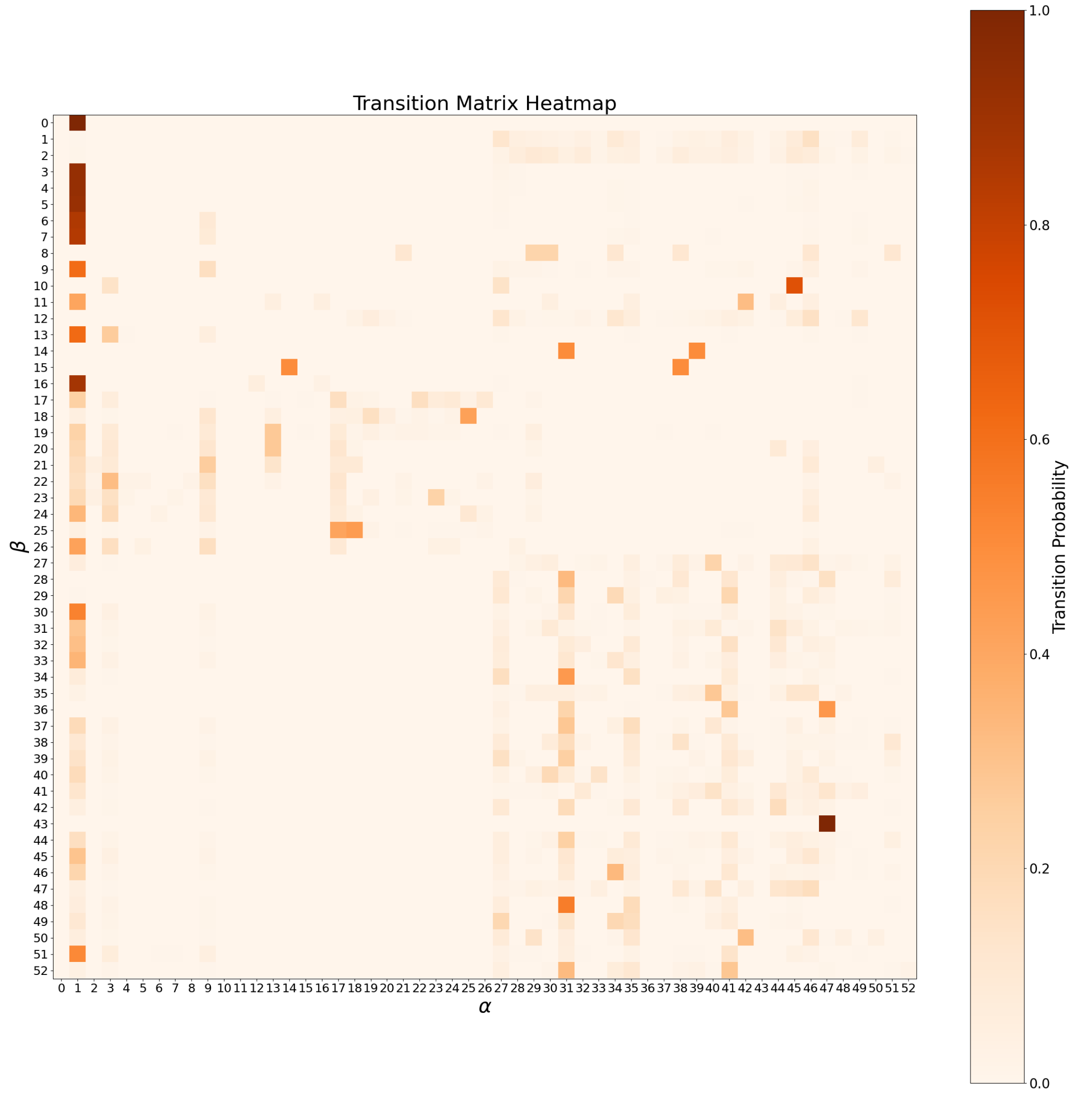


Figure 11: Hinton diagram of transition matrix of the English language, defined via War and Peace by Tolstoy.

5.b

The latent variables $\sigma(s)$ for different s are not independent. As specified in the beginning of the exercise, the mapping between symbols is one to one. Thus, if $\sigma(a) = s$, then the rest of the symbols $\sigma(s_i) \neq s$ cannot be s.
Given that we are approximating the non-encrypted symbols with a first-order Markov chain as specified in equation 55, we can use such approximation for the encrypted symbols as well. Furthermore, the mapping from non-encrypted symbols to encrypted symbols is one to one, this

allows us to use the same transition matrix specified in equation 55.

$$P(e_1, ..., e_n, \sigma) = P(e_1, \sigma) \prod_{i=1}^{n} P(e_i, \sigma \mid e_{i-1}, \sigma) = P(\sigma^{-1}(e_1)) \prod_{i=1}^{n} P(\sigma^{-1}(e_i) \mid \sigma^{-1}(s_{e-1})) \tag{61}$$

$$= P(\sigma^{-1}(e_1)) \prod_{\alpha,\beta} \psi(\alpha = \sigma^{-1}(e_i), \beta = \sigma^{-1}(e_{i-1})) \tag{62}$$

Where I have tried to express σ^{-1} as the function that decrypts the encrypted symbol e_i.

5.c

Using the proposal given in the exercise, we are choosing 2 different symbols from the 53 characters long string in "symbols.txt". But once the first symbol is chosen, the second can be any other symbol except for that one. For lack of better words, we are grabbing two balls from a bag containing 53 different balls. Thus, the probability of our proposal is given by $S(\sigma \to \sigma') = \frac{1}{\binom{2}{53}}$ $= \frac{2}{53 \times 52}$.
Given that our proposal is symmetric ($S(\sigma \to \sigma') = S(\sigma' \to \sigma)$) we can express our rejection kernel as:

$$A(\sigma \mid \sigma') = \min \left\{ 1, \frac{P(e_1, ..., e_n, \sigma')}{P(e_1, ..., e_n, \sigma)} \right\} \tag{63}$$

$$= \frac{P\left((\sigma')^{-1}(e_1)\right) \prod_{i=2}^{n} \psi\left((\sigma')^{-1}(e_i), (\sigma')^{-1}(e_{i-1})\right)}{P\left(\sigma^{-1}(e_1)\right) \prod_{i=2}^{n} \psi\left(\sigma^{-1}(e_i), \sigma^{-1}(e_{i-1})\right)} \tag{64}$$

5.d

I attempt to describe what I implemented with words.
We first obtain the transition matrix for the English language via a large enough text. In our case this large enough text is War and Peace by Tolstoy translated to English. With such transition matrix, we can calculate the probability for any sequence of symbols to be the English language. Using such probability, we propose different σ' in order to return different decryptions of the encrypted text. If such decrypted text has more chance to be the English language, that means our proposed σ' did a better job at decrypting the message, allowing us to keep it. Iterate through the algorithm until we hopefully find the appropriate σ.

I attach my results:

```
iteration: 0
[:p1lpl]x:r?!pw:np1]!?pmx;:?!w.;?pl?w!gp1lp/ws-?!prwm?p1?pg]

iteration: 100
v.udlulfe.pjouc.nudfojugea.joc:ajuljcomudluxcsbjoupcgjudjumf

iteration: 200
v.urlul e.pyouc.mur oyugea.yocbayulyconurlufcshyoupcgyuryun

iteration: 300
,murcuclempyou m.urloyugeamyo dayucy oturcuf shyoup gyuryutl
```

```
iteration: 400
,marcaciempyua m.ariuyageomyu doyacy utarcaf shyuap gyaryati

iteration: 500
lmarfafiemh uanm.ariu ageom undo af nutarfacns, uahng ar ati

iteration: 600
rmalfafiem. naumpalin ageom nudo af untalfacus, na.ug al ati

iteration: 700
rmalfafiem. naumpalin ageom nudo af untalfacus, na.ug al ati

iteration: 800
rmalfafiom. naumpalin agoem nude af unsalfacut, na.ug al asi

iteration: 900
rmalfafiom. naumpalin agoem nude af unsalfacut, na.ug al asi

iteration: 1000
rmalwawiom. naumpalin agoem nude aw unsalwacut, na.ug al asi

iteration: 1100
rsalwawios. nauspalin agoes nude aw unmalwacut, na.ug al ami

iteration: 1200
rsalwawios. nauspalin agoes nude aw unmalwacuth na.ug al ami

iteration: 1300
rsalyayuos. naispalun agoes nide ay inmalyacith na.ig al amu

iteration: 1400
rsalyayous. naispalon agues nife ay inmalyacith na.ig al amo

iteration: 1500
rsalyayous. naispalon agues nife ay inmalyacith na.ig al amo

iteration: 1600
rsalyayous. naispalon agues nife ay inmalyacith na.ig al amo

iteration: 1700
rsalyayous. naispalon agues nife ay inmalyacith na.ig al amo

iteration: 1800
rsalyayous. naispalon agues nife ay inmalyacith na.ig al amo

iteration: 1900
rsalyayous. naispalon agues nife ay inmalyacith na.ig al amo
```

```
iteration: 2000
rsalyayous. naispalon agues nife ay inmalyacith na.ig al amo

iteration: 2100
rm ly youm.an imp lona guemanifea yains ly cithan .iga la so

iteration: 2200
em ly youm.an imp lona gurmanifra yains ly cithan .iga la so

iteration: 2300
ar py your.en irl pone gumrenifme yeins py kithen .ige pe so

iteration: 2400
an py youn.er inm pore gulnerifle yeirs py kither .ige pe so

iteration: 2500
an py youn.er inm pore gulnerifle yeirs py kither .ige pe so

iteration: 2600
an py youn.er inm pore gulnerifle yeirs py kither .ige pe so

iteration: 2700
an py youn.er inm pore gulnerifle yeirs py kither .ige pe so

iteration: 2800
in py youn.er anm pore gulnerafle years py kather .age pe so

iteration: 2900
in py youn.er anm pore gulnerafle years py kather .age pe so

iteration: 3000
in py youn.er anm pore gulnerafle years py kather .age pe so

iteration: 3100
in py youn.er anm pore gulnerafle years py kather .age pe so

iteration: 3200
in py youn.er ang pore mulnerafle years py kather .ame pe so

iteration: 3300
in py youn.er ang pore mulnerafle years py kather .ame pe so

iteration: 3400
in my youn.er ang more pulnerafle years my kather .ape me so

iteration: 3500
in my youn.er ang more pulnerafle years my kather .ape me so
```

```
iteration: 3600
in my youn.er ang more pulnerafle years my kather .ape me so

iteration: 3700
in my youn.er ang more pulnerakle years my father .ape me so

iteration: 3800
in my youn.er ang more pulneravle years my father .ape me so

iteration: 3900
in my younker ang more dulneravle years my father kade me so

iteration: 4000
in my younker ang more dulneravle years my father kade me so

iteration: 4100
in my younker ang more dulneravle years my father kade me so

iteration: 4200
in my younker ang more dulneravle years my father kade me so

iteration: 4300
in my younker ang more dulneravle years my father kade me so

iteration: 4400
in my younker ang more dulneravle years my father kade me so

iteration: 4500
in my younker ang more dulneravle years my father kade me so

iteration: 4600
in my younker ang more dulneravle years my father kade me so

iteration: 4700
in my younker ang more dulneravle years my father kade me so

iteration: 4800
in my younker ang more dulneravle years my father kade me so

iteration: 4900
in my younker ang more dulneravle years my father kade me so

iteration: 5000
in my younker ang more dulneravle years my father kade me so

iteration: 5100
in my younker ang more dulneravle years my father kade me so
```

```
iteration: 5200
in my younker and more gulneravle years my father kage me so

iteration: 5300
in my younker and more gulneravle years my father kage me so

iteration: 5400
in my younker and more gulneravle years my father kage me so

iteration: 5500
in my younker and more gulneravle years my father kage me so

iteration: 5600
in my younker and more gulneravle years my father kage me so

iteration: 5700
in my younker and more gulneravle years my father kage me so

iteration: 5800
in my younger and more kulneravle years my father gake me so

iteration: 5900
in my younger and more kulneravle years my father gake me so

iteration: 6000
in my younger and more kulneravle years my father gake me so

iteration: 6100
in my younger and more kulneravle years my father gake me so

iteration: 6200
in my younger and more kulnerable years my father gake me so

iteration: 6300
in my younger and more kulnerable years my father gake me so

iteration: 6400
in my younger and more kulnerable years my father gake me so

iteration: 6500
in my younger and more kulnerable years my father gake me so

iteration: 6600
in my younger and more kulnerable years my father gake me so

iteration: 6700
in my younger and more kulnerable years my father gake me so
```

```
iteration: 6800
in my younger and more kulnerable years my father gake me so

iteration: 6900
in my younger and more kulnerable years my father gake me so

iteration: 7000
in my younger and more kulnerable years my father gake me so

iteration: 7100
in my younger and more kulnerable years my father gake me so

iteration: 7200
in my younger and more kulnerable years my father gake me so

iteration: 7300
in my younger and more kulnerable years my father gake me so

iteration: 7400
in my younger and more kulnerable years my father gake me so

iteration: 7500
in my younger and more kulnerable years my father gake me so

iteration: 7600
in my younger and more kulnerable years my father gake me so

iteration: 7700
in my younger and more kulnerable years my father gake me so

iteration: 7800
in my younger and more vulnerable years my father gave me so

iteration: 7900
in my younger and more vulnerable years my father gave me so

iteration: 8000
in my younger and more vulnerable years my father gave me so
```

The text did not change anymore after iteration 8000.

We can also plot how the likelihood is maximised as iterations increase:

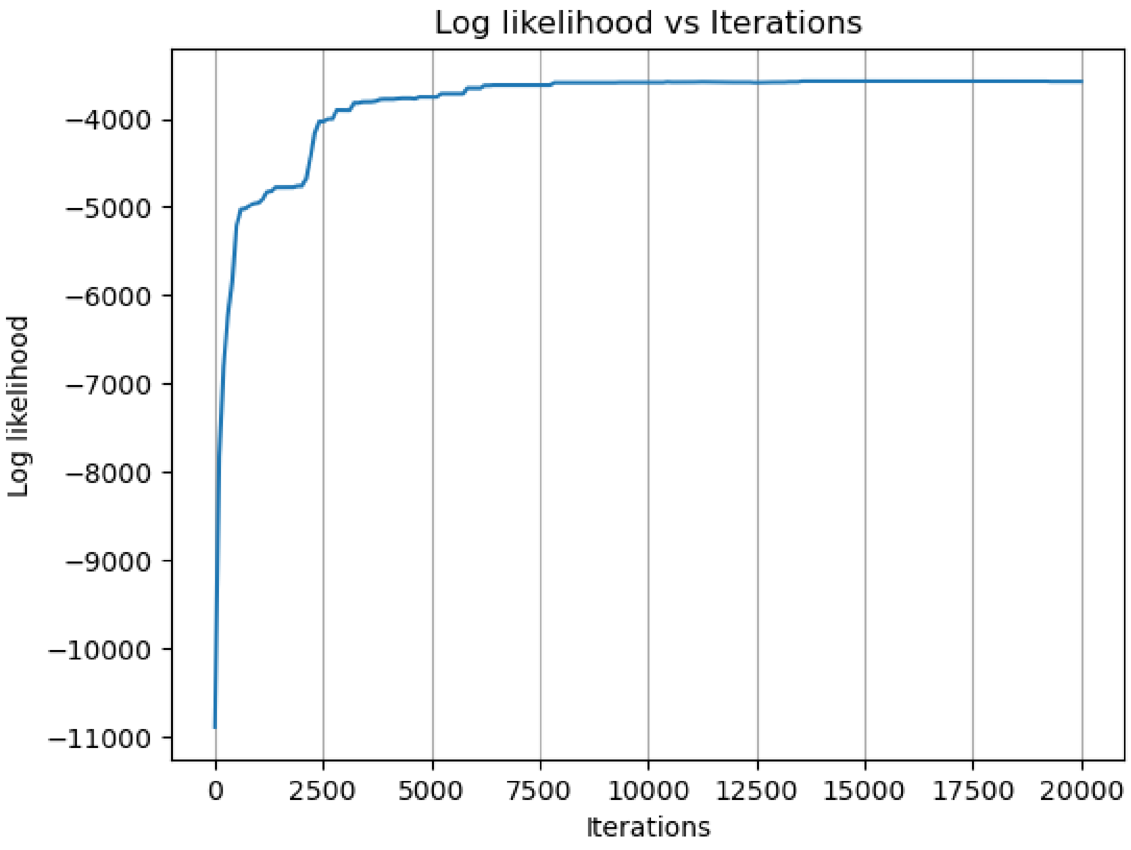


Figure 12: We could infer from the graph that burn-in settles shortly after the 7500-th iteration, which coincides with the iteration we no longer see the decrypted text improve!

Note: I have decided to show one of the lengthiest convergences I have managed to acquire to show what the “worst-case scenario” might be. When running my code, you should converge with less iterations.

Although my algorithm converges to the desired result the majority of the times, I have noticed that it is not uncommon for it to converge to what I am going to describe as local minima. This could be improved by “cleaning” our transition matrix or improving initialisation conditions. Taking into account the considerable burn-in shown in figure 12, it is probably the latter that can be improved the most. The transition matrix could be improved by cleaning the large text (War and Peace by Tolstoy) out of symbols that may not be in the symbols.txt provided file.

The initialisation conditions are harder to improve as they will always be motivated by some (potentially wrong) bias. How I would improve our initialisation conditions is by first studying the likelihood changing only the most popular symbols in the English language. For example, the blank space. We could first study how our likelihood changes by considering 53 different σ' only, each one having $\sigma(blankspace) = s$ different symbol s. Determine which σ returns the biggest likelihood and start with such σ. Do this for the blank space, the letter e and the letter t (three most used characters in the English language according to Slide 103 of the Notes), and initialise the algorithm with sigma that maximises the likelihood for these 3 symbols.

Why the most popular characters?
I have found that when my algorithm gets stuck in a local minima, it is hugely in part because it has proposed a σ' that has $\sigma(blankspace) = s$ that is not the desired symbol s, but is good enough to maximise likelihood enough to never be able to exit it afterwards. This makes sense, since "blankspace" is the most used charachter in the English language, small changes in σ for such symbol can cause big effects on the likelihood.
Note: I have not implemented such initialisation in my code. I have attempted to initialise with what I thought might be the "worse" initialisation. My initialisation supposes that the encryption is no encryption at all.

Why would I attempt to implement the "worse" initialisation?
I attempted to test the algorithm's robustness by proving its convergence even if we were unlucky enough to initialise with perhaps one of the worst initialisations possible.

5.e

A Markov chain is ergodic if it's possible to go from any state to any state (not necessarily in one move). Evidently, having some values in our transition matrix $\psi(\alpha, \beta)$ be zero could suppose our Markov chain to not be ergodic. In other words, zero values in our transition matrix may suppose never being able to reach certain states due to our (possibly unfortunate) initialisation. We can restore ergodicity to the chain by adding a small value to each element of the transition matrix. This is done in my code.

5.f

Symbol probabilities alone would not suffice. Symbol probabilities alone do not take into account any type of order. Thus, one can think about this as a Microstate and Macrostate problem. Given any sentence: "i am enric" (all in lower case to make my point easier), we can consider as a Macrostate the symbols that construct such sentence: 2 spaces, two i, one m, one a, etc... We can consider a Microstate the ways in which such letters can be arranged to return the same Macrostate. By using symbol probabilities alone, our model would consider equally likely the following two sentences: "i am enric" and "ecmianri " as both contain the same symbols. Evidently, the amount of possible Microstates becomes huge as sentence length increases. Making the possibility for our model to return the "ground truth" Microstate (sentence) to be close to 0.

Although a second order Markov Chain might yield better results than our first order approximation, our transition matrix would become a 3-Dimensional matrix, thus increasing the complexity of acquiring it and using it. A second order Markov Chain might be worth considering if the first order approximation did not produce satisfactory results. However, given the fact that our first order approximation has already been successful in decrypting the text, one could consider a second order approximation to thus be inefficient in our case.

If encryption were not to be one to one, then our model would not work. Our log probability function is built on the assumption that encryption is bijective. If that is not the case, then our log probability function and thus the rejection kernel would be faulty.

If we were to acquire the transition matrix for the Chinese language, most of its entries would be zero or close to zero and its dimensions (amount of rows and columns) would be huge compared to the English language matrix. The combination of these two things would result in our model not working. There are states with a chance for us to get to so small that given finite time and iterations (MCMC), one could consider impossible for our model to ever visit.

6 Exercise 6

7 Exercise 7

7.a

We are given: $f(x, y) = x + 2y$ constraint by $y^2 + xy = 1$. We proceed by using Lagrangian multipliers.

$$\text{Lagrange Multipliers} \begin{cases} \boldsymbol{\nabla} f(x,y) & = \lambda \boldsymbol{\nabla} g(x,y) \\ y^2 + xy & = 1 \end{cases} \tag{65}$$

We compute each gradient respectively:

$$f_x = 1 \tag{66}$$
$$f_y = 2 \tag{67}$$
$$g_x = y \tag{68}$$
$$g_y = 2y + x \tag{69}$$

We thus acquire 3 equations for our 3 unknowns:

$$1 = \lambda y \tag{70}$$
$$2 = \lambda(2y + x) \tag{71}$$
$$y^2 + xy = 1 \tag{72}$$

We can acquire $y = \frac{1}{\lambda}$ directly from equation 70 and substitute it to 71 to acquire:

$$2 = \lambda(\frac{2}{\lambda} + x) \tag{73}$$
$$x = \frac{2}{\lambda} - \frac{2}{\lambda} \tag{74}$$
$$x = 0 \tag{75}$$

Substitute 75 to 72 to acquire:

$$y^2 = 1 \tag{76}$$
$$y = \pm 1 \tag{77}$$

Finally, substituting 76 in 70, one can acquire the Lagrange multiplier to be $\lambda = \pm 1$.
Thus acquiring the following solution:

$$\text{Points where extrema occur} \begin{cases} (x = 0, y = 1) & \rightarrow \lambda = 1 \\ (x = 0, y = -1) & \rightarrow \lambda = -1 \end{cases} \tag{78}$$

7.b

Need to find a function $F(x)$ that fulfills $F(\ln(a)) = 0$. In order to then apply Newton's method and find the roots. Such desired function is:

$$F(x) = e^x - a \tag{79}$$

Thus when $F(\ln(a)) = e^{\ln(a)} - a = a - a = 0$. Applying slide 257/287 from the Notes (slide 252 if you follow slide page counter), we use Newton's method to specify an update equation for this problem.

$$x_{n+1} = x_n - \frac{F(x_n)}{F'(x_n)} \tag{80}$$

$$x_{n+1} = x_n - \frac{e^{x_n} - a}{e^{x_n}} \tag{81}$$

Was not asked in the summative assignment to prove that above algorithm converges but I proceed to show such result just in case: Newton's algorithm converges if and only if: $F(x)F'(x) < (F'(x))^2$.

$$(e^x - a)e^x < e^{2x} \tag{82}$$

$$e^{2x} - ae^x < e^{2x} \tag{83}$$

Thus, for positive a only ($a > 0$), this algorithm converges. This makes sense when we contextualise the result with our original desire to approximate $\ln(a)$.

8 Exercise 8

8.a

As the hint suggests, we propose set $\mathcal{S}$ to study $R_A(\mathbf{x})$ as we know such set is compact. We denote $\mathbf{y}$ for any $\mathbb{R}^n$vector that is also in set $\mathcal{S}$: $\mathbf{y} \in \mathcal{S}$. In other words, any vector $\mathbf{y}$ that is unitary.

$$R_A(\mathbf{y}) = \frac{\mathbf{y}^T A\mathbf{y}}{\|\mathbf{y}\|^2} = \mathbf{y}^T A\mathbf{y} = q_A(\mathbf{y}) \tag{84}$$

Reminder: $\|\mathbf{y}\| = 1$. Thanks to the extreme value Theorem of calculus, we know a continuous function on a compact domain attains its maximum and minimum. We are given $q_A(\mathbf{y})$ is always continuous. We are given $\mathcal{S}$ is compact. Thus $\sup_{\mathbf{y}\in\mathcal{S}} q_A(\mathbf{y})$ is attained. We have shown in equation 84 that $R_A(\mathbf{y}) = q_A(\mathbf{y})$. Thus, $\sup_{\mathbf{y}\in\mathcal{S}} R_A(\mathbf{y})$ is attained.

Finally, we realise the following for any $\mathbf{x} \in \mathbb{R}^n$.

$$R_A(\mathbf{x}) = \frac{\mathbf{x}^T A\mathbf{x}}{\|\mathbf{x}\|^2} = \frac{\mathbf{x}}{\|\mathbf{x}\|} A \frac{\mathbf{x}}{\|\mathbf{x}\|} = \mathbf{y}A\mathbf{y} = R_A(\mathbf{y}) \tag{85}$$

Reminder: $\mathcal{S} = \{\mathbf{y} \in \mathbb{R}^n \mid \|\mathbf{y}\| = 1\}$.Thus $\frac{\mathbf{x}}{\|\mathbf{x}\|} \in \mathcal{S}$.
We have shown above $\sup_{\mathbf{y}\in\mathcal{S}} R_A(\mathbf{y})$ is attained. We have shown $R_A(\mathbf{x}) = R_A(\mathbf{y})$ for any $\mathbf{x} \in \mathbb{R}^n$. Thus, $\sup_{\mathbf{x}\in\mathbb{R}^n} R_A(\mathbf{x})$ is attained.

8.b

Substituting $x = \sum_{i=1}^n (\xi_i^T x)\xi_i$ into $R_A(x)$ yields:

$$R_A(x) = \frac{\left(\sum_{i=1}^n (\xi_i^T x)\xi_i\right)^T A \left(\sum_{i=1}^n (\xi_i^T x)\xi_i\right)}{\left(\sum_{i=1}^n (\xi_i^T x)\xi_i\right)^T \left(\sum_{i=1}^n (\xi_i^T x)\xi_i\right)} \tag{86}$$

Recognise $(\xi_i^T x)$ is a scalar.

$$R_A(x) = \frac{\left(\sum_{i=1}^n (\xi_i^T x)\xi_i\right)^T \left(\sum_{i=1}^n (\xi_i^T x)A\xi_i\right)}{\left(\sum_{i=1}^n (\xi_i^T x)\xi_i\right)^T \left(\sum_{i=1}^n (\xi_i^T x)\xi_i\right)} \tag{87}$$

Recognise $A\xi_i = \lambda_i\xi_i$

$$R_A(x) = \frac{\sum_{i=1}^n \lambda_i \left((\xi_i^T x)\xi_i\right)^T \left((\xi_i^T x)\xi_i\right)}{\sum_{i=1}^n \left((\xi_i^T x)\xi_i\right)^T \left((\xi_i^T x)\xi_i\right)} \tag{88}$$

Given that $\lambda_1 \geq \lambda_2 \geq \lambda_3 ...$

$$R_A(x) = \frac{\sum_{i=1}^n \lambda_i \left((\xi_i^T x)\xi_i\right)^T \left((\xi_i^T x)\xi_i\right)}{\sum_{i=1}^n \left((\xi_i^T x)\xi_i\right)^T \left((\xi_i^T x)\xi_i\right)} \leq \frac{\sum_{i=1}^n \lambda_1 \left((\xi_i^T x)\xi_i\right)^T \left((\xi_i^T x)\xi_i\right)}{\sum_{i=1}^n \left((\xi_i^T x)\xi_i\right)^T \left((\xi_i^T x)\xi_i\right)} \tag{89}$$

$$= \lambda_1 \frac{\cancel{\sum_{i=1}^n \left((\xi_i^T x)\xi_i\right)^T \left((\xi_i^T x)\xi_i\right)}}{\cancel{\sum_{i=1}^n \left((\xi_i^T x)\xi_i\right)^T \left((\xi_i^T x)\xi_i\right)}} \tag{90}$$

$$= \lambda_1 \tag{91}$$

Thus, we have shown $R_A(x) \leq \lambda_1$.

8.c

We can re-use the expressions we acquired in the above subsection, where instead of smaller or equal to, we can use smaller than strictly due to ($\lambda_1 > \lambda_{k+1}, ..., \lambda_n$):

$$R_A(x) = \frac{\sum_{i=1}^{n} \lambda_i \left((\xi_i^T x)\xi_i\right)^T \left((\xi_i^T x)\xi_i\right)}{\sum_{i=1}^{n} \left((\xi_i^T x)\xi_i\right)^T \left((\xi_i^T x)\xi_i\right)} < \frac{\sum_{i=1}^{n} \lambda_1 \left((\xi_i^T x)\xi_i\right)^T \left((\xi_i^T x)\xi_i\right)}{\sum_{i=1}^{n} \left((\xi_i^T x)\xi_i\right)^T \left((\xi_i^T x)\xi_i\right)} = \lambda_1 \tag{92}$$

Where in the last step we did the same as in above subsection.
I forgot to mention, there are $\xi_{k+1}x, ..., \xi_n x$ that are not zero, thus above holds. Thus $R_A(x) < \lambda_1$