Unsupervised Learning (COMP0086) - Coursework 1

Student numbers: 21146187

November 22, 2021

1. Models for binary vectors

(a)

Multivariate Gaussian distributions models probability densities over multiple variables, each of which can take values on the real line. A binary set of images contains multiple variables that can each take only discrete variables (black or white), hence modelling this dataset by a multivariate Gaussian would be inappropriate.

(b) ML estimate of p

We denote the data set $\left\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\right\}$ by $\mathcal{D}$

$$\begin{aligned}\mathsf{P}(\mathcal{D}|\mathbf{p}) &= \prod_{i=1}^{N} \mathsf{P}(\mathbf{x}^{(i)}|\mathbf{p}) \\ &= \prod_{i=1}^{N}\prod_{d=1}^{D} p_d^{x_d^{(i)}}(1-p_d)^{(1-x_d^{(i)})}\end{aligned}$$

Maximum likelihood $\mathbf{p}$ is found by taking the logarithm of the probability above, and finding the maximising arguments.

$$\mathbf{p}^{\text{MLE}} = \text{argmax}\left(\sum_{i=1}^{N}\sum_{d=1}^{D}\left[x_d^{(i)}\ln(p_d) + (1-x_d^{(i)})\ln(1-p_d)\right]\right)$$

We solve for the gradient of the likelihood equals 0, wrt $\mathbf{p}$.

$$\begin{aligned}0 &= \vec{\boldsymbol{\nabla}}_{\mathbf{p}} \sum_{d=1}^{D}\left[\left(\sum_{i=1}^{N} x_d^{(i)}\right)\ln(p_d) + (N - \left(\sum_{i=1}^{N} x_d^{(i)}\right))\ln(1-p_d)\right] \\ 0 &= \frac{1}{p_d}\left(\sum_{i=1}^{N} x_d^{(i)}\right) - \frac{1}{1-p_d}\left(1 - \left(\sum_{i=1}^{N} x_d^{(i)}\right)\right) \\ \mathbf{p}^{\text{MLE}} &= \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}^{(i)} = \hat{\mathbf{x}}\end{aligned}$$

We find that the maximum likelihood estimate of the parameter vector is the mean of all the images.

MAP estimate of p

Assuming beta priors, we want to find the parameters that maximise $\mathsf{P}(\mathbf{p}|\mathcal{D})$. We can use

$$\mathsf{P}(\mathbf{p}|\mathcal{D}) = \frac{\mathsf{P}(\mathcal{D}|\mathbf{p})\mathsf{P}(\mathbf{p})}{\mathsf{P}(\mathcal{D})}$$

$$\ln(\mathsf{P}(\mathbf{p}|\mathcal{D})) = \ln(\mathsf{P}(\mathcal{D}|\mathbf{p})\mathsf{P}(\mathbf{p})) + \text{constant}$$

Then we can write

$$\begin{aligned}
\mathbf{p}^{MAP} &= \text{argmax}\left(\ln\left(\mathsf{P}(\mathcal{D}|\mathbf{p})\mathsf{P}(\mathbf{p})\right)\right) \\
0 &= \vec{\boldsymbol{\nabla}}_{\mathbf{p}}\left(\ln\left(\prod_{i=1}^{N}\mathsf{P}(\mathbf{x}^{(i)}|\mathbf{p})\prod_{d=1}^{D}\mathsf{P}(p_d)\right)\right) \\
0 &= \vec{\boldsymbol{\nabla}}_{\mathbf{p}}\left(\ln\left(\prod_{i=1}^{N}\prod_{d=1}^{D}p_d^{x_d^{(i)}}(1-p_d)^{(1-x_d^{(i)})}\prod_{a=1}^{D}\frac{1}{B(\alpha,\beta)}p_a^{\alpha-1}(1-p_a)^{\beta-1}\right)\right) \\
0 &= \vec{\boldsymbol{\nabla}}_{\mathbf{p}}\left(\ln\left(\prod_{i=1}^{N}\prod_{d=1}^{D}p_d^{(x_d^{(i)}+\alpha-1)}(1-p_d)^{(\beta-x_d^{(i)})}\right)\right) \\
0 &= \vec{\boldsymbol{\nabla}}_{\mathbf{p}}\left(\sum_{i=1}^{N}\sum_{d=1}^{D}\left[(x_d^{(i)}+\alpha-1)\ln(p_d) + (\beta-x_d^{(i)})\ln(1-p_d)\right]\right)
\end{aligned}$$

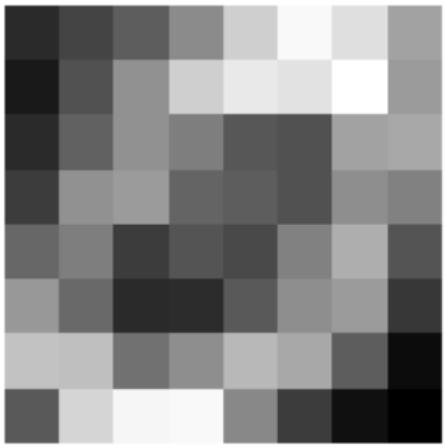
We define $\hat{N}_d = \sum_{i=1}^{N} x_d^{(i)}$, so the above becomes

$$\begin{aligned}
0 &= \vec{\boldsymbol{\nabla}}_{\mathbf{p}}\left(\sum_{d=1}^{D}\left[(\hat{N}_d+(\alpha-1))\ln(p_d) + (\beta N-\hat{N}_d)\ln(1-p_d)\right]\right) \\
0 &= \frac{1}{p_d}(\hat{N}_d+(\alpha-1)) - \frac{1}{1-p_d}(\beta N-\hat{N}_d) \\
p_d &= \frac{\hat{x}_d+\alpha-1}{\alpha+\beta-1}
\end{aligned}$$

So we find $p_d^{MAP} = \frac{\hat{x}_d + \alpha - 1}{\alpha + \beta - 1}$, where $\hat{x}_d = \frac{\hat{N}_d}{N}$

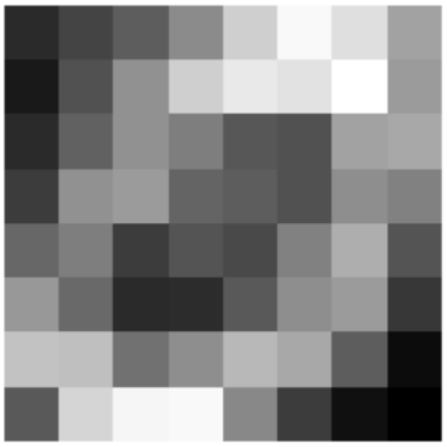
(d) and (e)

```
[0.13 0.21 0.29 0.43 0.64 0.77 0.69 0.5  0.08 0.25 0.45 0.64 0.72 0.7
 0.79 0.48 0.13 0.3  0.45 0.39 0.27 0.25 0.5  0.52 0.19 0.45 0.48 0.31
 0.29 0.25 0.44 0.4  0.32 0.39 0.19 0.26 0.23 0.4  0.54 0.26 0.47 0.33
 0.13 0.14 0.28 0.44 0.48 0.17 0.6  0.59 0.35 0.44 0.57 0.52 0.29 0.04
 0.28 0.66 0.76 0.77 0.42 0.19 0.05 0.  ]
```



(a) a

```
[0.426 0.442 0.458 0.486 0.528 0.554 0.538 0.5   0.416 0.45  0.49  0.528
 0.544 0.54  0.558 0.496 0.426 0.46  0.49  0.478 0.454 0.45  0.5   0.504
 0.438 0.49  0.496 0.462 0.458 0.45  0.488 0.48  0.464 0.478 0.438 0.452
 0.446 0.48  0.508 0.452 0.494 0.466 0.426 0.428 0.456 0.488 0.496 0.434
 0.52  0.518 0.47  0.488 0.514 0.504 0.458 0.408 0.456 0.532 0.552 0.554
 0.484 0.438 0.41  0.4  ]
```



(b) b

Figure 1: 6a MLE estimate of $\mathbf{p}$ in explicit and visual form. 6b MAP estimate of $\mathbf{p}$ in explicit and visual form.

Whether the MAP estimate is better or worse than the ML estimate depends on

how accurate the priors imposed on $\mathbf{p}$ are.

```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.distributions as dist
%matplotlib inline
```

```
Y = np.loadtxt('binarydigits.txt')
N, D = Y.shape
```

```
"""
Part (d) function to learn the ML parameters of a multivariate Bernoulli for the data set,
and display them as a grayscale image
"""
def learn_MLp(Y):
    N, D = Y.shape
    p = np.sum(Y,axis=0)/N
    print(p)
    plt.figure()
    plt.imshow(np.reshape(p, (8,8)),
               interpolation="None",
               cmap='gray')
    plt.axis('off')
    plt.show()

learn_MLp(Y)
```

```
"""
part (e) Function that learns the MAP parameters with alpha and beta = 3,
and displays the new parameters as an image
"""
def learn_MAPp(Y,alpha=3,beta=3):
    N, D = Y.shape
    p = np.sum(Y,axis=0)/N
    p += (alpha-1)
    p = p/(beta+alpha-1)
    print(p)
    plt.figure()
    plt.imshow(np.reshape(p, (8,8)),
               interpolation="None",
               cmap='gray')
    plt.axis('off')
    plt.show()

learn_MAPp(Y)
```

Figure 2: Code for learning the ML and MAP estimates

2. Model Selection

For a set of Bernoulli models $\mathcal{M}_{\{i\in a,b,c\}}$ with equal prior probability and associated parameters $\mathbf{p}^{(i)}$, we wish to find the probabilities $P(\mathcal{M}_i|\mathcal{D})$ relative to other models ($\mathcal{D}$ is the data). Using Bayes rule:

$$P(\mathcal{M}_i|\mathcal{D}) = \frac{P(\mathcal{D}|\mathcal{M}_i)P(\mathcal{M}_i)}{P(\mathcal{D})} \propto P(\mathcal{D}|\mathcal{M}_i)$$

We are only concerned with the relative probabilities, so we see it suffices to find the likelihood. This is found by marginalising over the parameters of the model, as below

$$P(\mathcal{D}|\mathcal{M}_i) = \int d\mathbf{p}^{(i)} P(\mathcal{D}|\mathbf{p}^{(i)}, \mathcal{M}_i) P(\mathbf{p}^{(i)}|\mathcal{M}_i)$$

Model (a)

$\mathcal{M}_a$: $p_d^{(a)} = \frac{1}{2}\ \forall d \in \{1, \ldots, D\}$

Hence each parameter component has a dirac-delta probability distribution $P(p_d^{(a)}) = \delta(p_d^{(a)} - \frac{1}{2})$. It can easily be seen that $P(\mathcal{D}|p_d^{(a)}) = \prod_{d=1}^{N} D\left(\frac{1}{2}\right)$. Now we can write the likelihood:

$$\begin{aligned} P(\mathcal{D}|\mathcal{M}_a) &= \prod_{d=1}^{D} \int_0^1 dp_d^{(a)} \left(\frac{1}{2}\right)^N \delta\left(p_d^{(a)} - \frac{1}{2}\right) \\ &= \left(\frac{1}{2}\right)^{ND} \end{aligned}$$

Model (b)

$\mathcal{M}_b$: $p_d^{(b)} = p\ \forall d \in \{1, \ldots, D\}$

As instructed in the question, we assume a uniform prior on the unknown parameter p. By substituting $p_d = p\ \forall d$ in the Bernoulli formula, we find the probability of the data given the parameters:

$$\begin{aligned} P(\mathbf{x}^{(i)}|\mathbf{p}^{(b)}) &= \prod_{d=1}^{D} p^{x_d^{(i)}} (1-p)^{(1-x_d^{(i)})} \\ &= p^{\sum_{d=1}^{D} x_d^{(i)}} (1-p)^{(D - \sum_{d=1}^{D} x_d^{(i)})} \\ \Rightarrow P(\mathcal{D}|\mathbf{p}^{(b)}) &= p^{\sum_{i=1}^{N} \sum_{d=1}^{D} x_d^{(i)}} (1-p)^{(ND - \sum_{i=1}^{N} \sum_{d=1}^{D} x_d^{(i)})} \end{aligned}$$

For convenience, we define $t = \sum_{i=1}^{N} \sum_{d=1}^{D} x_d^{(i)}$. Then the likelihood is

$$
\begin{aligned}
P(\mathcal{D}|\mathcal{M}_b) &= \int d\mathbf{p}^{(b)} P(\mathcal{D}|\mathbf{p}^{(b)}) P(\mathbf{p}^{(b)}) \\
&= \int_0^1 p^t (1-p)^{(ND-t)} dp \\
&= \int_0^1 p^{(t+1)-1} (1-p)^{(ND-t+1)-1} dp \\
&= B(\alpha^{(b)}, \beta^{(b)})
\end{aligned}
$$

where $\alpha^{(b)} = 1 + \sum_{i=1}^{N} \sum_{d=1}^{D} x_d^{(i)}$ and $\beta^{(b)} = 1 + ND - \sum_{i=1}^{N} \sum_{d=1}^{D} x_d^{(i)}$ and B is the Beta function.

Model (c)

$\mathcal{M}_c$: $p_d^{(c)} \sim \text{Uniform}[0,1]\ \forall d$

By the Bernoulli distribution:

$$
\begin{aligned}
P(\mathcal{D}|\mathbf{p}^{(c)}) &= \prod_{i=1}^{N} \prod_{d=1}^{D} p_d^{x_d^{(i)}} (1-p_d)^{(1-x_d^{(i)})} \\
\Rightarrow P(\mathcal{D}|\mathcal{M}_c) &= \int d\mathbf{p}^{(c)} \prod_{i=1}^{N} \prod_{d=1}^{D} p_d^{x_d^{(i)}} (1-p_d)^{(1-x_d^{(i)})} \\
&= \prod_{d=1}^{D} \left(\int_0^1 p_d^{\sum_{i=1}^{N} x_d^{(i)}} (1-p_d)^{(N-\sum_{i=1}^{N} x_d^{(i)})} dp_d \right) \\
&= \prod_{d=1}^{D} B(\alpha_d^{(c)}, \beta_d^{(c)})
\end{aligned}
$$

where $\alpha_d^{(c)} = 1 + \sum_{i=1}^{N} x_d^{(i)}$ and $\beta_d^{(c)} = 1 + N - \sum_{i=1}^{N} x_d^{(i)}$ and B is the Beta function. Therefore the ratios of the probabilities of models a:b:c are:

$$
\left(\frac{1}{2}\right)^{ND} : B(\alpha^{(b)}, \beta^{(b)}) : \prod_{d=1}^{D} B(\alpha_d^{(c)}, \beta_d^{(c)}) \tag{1}
$$

Code was run to calculate these proportions, the code is given in the appendix, the ratio calculated was:

$[7.486 \times 10^{-60} \ : \ 11741703 \ : \ 8.188 \times 10^{194}]$

3. EM for Binary Data

Apologies for the handwritten answers, I didn't have time to type up in Latex.

a) ~~Likelihood~~ Likelihood for a model consisting of K multivariate Bernoulli distributions

Let $P(\underline{x}^{(i)} | \underline{p}_k)$ denote the probability of ~~image~~ i the i^{th} image, given the Bernoulli parameter $\underline{p}_k$, ~~from~~ corresponding to the k^{th} Bernoulli distribution.

Let $\underline{\underline{P}} \in \mathbb{R}^{K \times D}$ be the matrix whose rows are the parameter vectors.

$\{\pi_1, \ldots, \pi_K\}$ $(0 \leq \pi_i \leq 1 ; \sum_k \pi_k = 1)$ denote the mixing proportions

$$P(\underline{x}^{(i)} | \underline{p}_k) = \prod_{d=1}^{D} p_{kd}^{x_d^{(i)}} (1 - p_{kd})^{(1 - x_d^{(i)})}$$

and

$$P(\underline{x}^{(i)} | \underline{\underline{P}}) = \sum_{k=1}^{K} \pi_k P(\underline{x}^{(i)} | \underline{p}_k)$$

$$= \sum_{k=1}^{K} \pi_k \prod_{d=1}$$

Let $D = \{\underline{x}^{(1)}, \ldots, \underline{x}^{(N)}\}$ denote the data

Images are i.i.d $\Rightarrow$

$$P(D | \underline{\underline{P}}) = \prod_{i=1}^{N} \left(\sum_{k=1}^{K} \pi_k \left(\prod_{d=1}^{D} p_{kd}^{x_d^{(i)}} (1 - p_{kd})^{x_d^{(i)}} \right) \right) \quad (1)$$

$\longrightarrow$

(1) (IS THE LIKELIHOOD)

c)

b) Responsibility of mixture component k for image n:

We introduce latent variable $s^{(n)} \in \{1, \ldots, K\}$

where $P(s^{(n)} = k \mid \underline{\pi}) = \pi_k$

Responsibilities:

$$r_{nk} = P(s^{(n)} = k \mid \underline{x}^{(n)}, \underline{\pi}, \underline{\underline{P}})$$

$$= \frac{P(\underline{x}^{(n)} \mid s^{(n)} = k, \underline{\underline{P}}, \underline{\pi}) P(s^{(n)} = k \mid \underline{\pi})}{P(\underline{x}^{(n)} \mid \underline{\underline{P}}, \underline{\pi})}$$

$$= \frac{P(\underline{x}^{(n)} \mid \underline{P}_k) \pi_k}{\sum_{k=1}^{K} P(\underline{x}^{(n)} \mid \underline{P}_k) \pi_k}$$

$$r_{nk} = \frac{\pi_k \prod_{d=1}^{D} p_{kd}^{x_d^{(n)}} (1 - p_{kd})^{(1 - x_d^{(n)})}}{\sum_{m=1}^{K} \pi_m \left(\prod_{d=1}^{D} p_{md}^{x_d^{(n)}} (1 - p_{md})^{(1 - x_d^{(n)})} \right)}$$

c) Maximising parameters for the expected log-joint

$$\underset{\underline{\pi}, \underline{\underline{P}}}{\operatorname{argmax}} \left(\sum_n \log\left(P\left(\underline{x}^{(n)}, s^{(n)} \mid \underline{\pi}, \underline{\underline{P}} \right) \right) \right)_{q(\{s^{(n)}\})}$$

~~We have~~

$$\mathcal{L} = \sum_{n=1}^{N} \log\left(P\left(\underline{x}^{(n)}, s^{(n)} \mid \underline{\pi}, \underline{\underline{P}} \right) \right)$$

$$= \sum_n \log$$

~~We introduce~~

$$\mathcal{L} = \sum_n \log\left(P\left(\underline{x}^{(n)} \mid s^{(n)}, \underline{\pi}, \underline{\underline{P}} \right) P\left(s^{(n)} \right) \right)$$

$$\langle \mathcal{L} \rangle_{q(\{s_n\})} = \sum_{n=1}^{N} \sum_{k=1}^{K} q\left(s^{(n)} = k \right) \log\left(P\left(\underline{x}^{(n)} \mid s^{(n)} = k, \underline{\pi}, \underline{\underline{P}} \right) P\left(s^{(n)} = k \right) \right)$$

$$r_{nk} \overset{\text{def}}{=} q\left(s^{(n)} = k \right)$$

$$\Rightarrow \langle \mathcal{L} \rangle_q = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \left[\log(\pi_k) + \log\left(P\left(\underline{x}^{(n)} \mid \underline{p}_k \right) \right) \right]$$

$$\langle \mathcal{L} \rangle_q = \sum_{n,k}^{N,K} r_{nk} \left[\log(\pi_k) + \sum_{d=1}^{D} \left(x_d^{(n)} \log(p_{kd}) + (1 - x_d^{(n)}) \log(1 - p_{kd}) \right) \right]$$

We want to find maximising parameters $\Rightarrow$
set ~~stationary~~ gradient w.r.t parameters to 0.

$$\nabla_{\underline{P}} \langle \mathcal{L} \rangle_q = 0$$

$$\frac{\partial}{\partial P_{\mu\nu}} \langle \mathcal{L} \rangle_q = 0$$

$$\frac{\partial}{\partial P_{\mu\nu}} \sum_{n=1}^{N} \sum_{k=1}^{K} \sum_{d=1}^{D} \left[r_{nk} x_d^{(n)} \log(P_{kd}) + r_{nk} (1 - x_d^{(n)}) \log(1 - P_{kd}) \right] = 0$$

$$\sum_{n} \sum_{k,d} \left[\frac{r_{nk} x_d^{(n)}}{P_{kd}} \delta_{\mu k} \delta_{\nu d} + \frac{r_{nk} (1 - x_d^{(n)})}{1 - P_{kd}} \delta_{\mu k} \delta_{\nu d} \right] = 0$$

$$\sum_{n=1}^{N} \left[\frac{r_{n\mu} x_\nu^{(n)}}{P_{\mu\nu}} - \frac{r_{n\mu} (1 - x_\nu^{(n)})}{1 - P_{\mu\nu}} \right] = 0$$

$$P_{\mu\nu} = \frac{\sum_{n=1}^{N} r_{n\mu} x_\nu^{(n)}}{\sum_{n=1}^{N} r_{n\mu}}$$

WH Smith SN3 3RX UK
Produced in China

We must maximise $\langle \mathcal{L} \rangle_Q$ w.r.t the mixing parameters under the constraint $\sum_K \pi_K = 1$

$$\Rightarrow \frac{\partial}{\partial \underline{\pi}} \left[\langle \mathcal{L} \rangle_Q - \lambda \left(\sum_{K=1}^{K} \pi_K - 1 \right) \right] = 0$$

$$\sum_{n=1}^{N} \frac{r_{nm}}{\pi_m} - \lambda = 0$$

$$\pi_m = \frac{1}{\lambda} \sum_{n=1}^{N} r_{nm}$$

Enforcing the constraint $\sum_m \pi_m = 1$

$$\Rightarrow \lambda = \sum_{n,m} r_{nm}$$

$$= \sum_n 1$$

$$= N$$

$$\Rightarrow \boxed{\pi_K = \frac{1}{N} \sum_{n=1}^{N} r_{nK}}$$

Part (d)

The EM algorithm was implemented for mixtures of K multivariate Bernoulli's for $K = \{2, 3, 4, 7, 10\}$. The log-likelihood is plotted over the iterations. The algorithm was run multiple times for different random initialisations of the parameter matrix P, the results for one of these initialisations is plotted below. (question (d) also requires me to display the parameters found, I have instead amalgamated this into question (e))

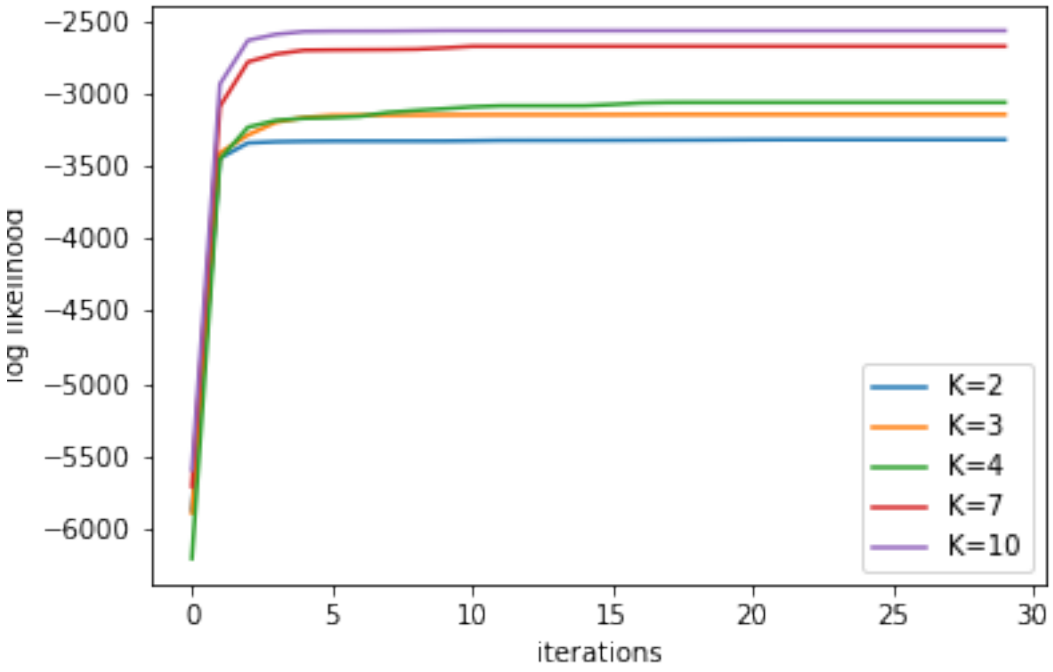


Figure 3: Log-likelihood for varying numbers of Bernoulli models against iteration of the EM algorithm

Part (e)

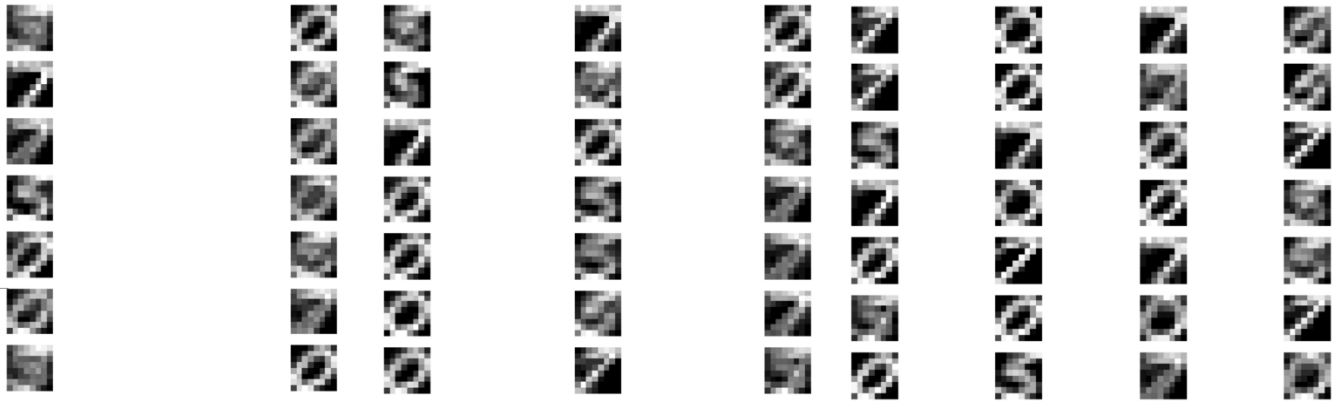
The algorithm was run a few times starting from randomly chosen initial conditions. The solutions are shown below, for each K the algorithm was run 7 times, to produce 7 different sets of learned parameters. Each K rows of the P matrix are displayed as K grayscale images, the π parameter is shown as a torch tensor. Explaining the figure below: Each K occupies a block in the figure, within each block there are 7

rows, corresponding to each random initialisation of the algorithm, and K columns, corresponding to each row of the P matrix found. The π vector found for each initialisation of the algorithm is displayed above the images (in the same order as the images).

```
tensor([0.6001, 0.3999], dtype=torch.float64)
tensor([0.1500, 0.8500], dtype=torch.float64)
tensor([0.2808, 0.7192], dtype=torch.float64)
tensor([0.1903, 0.8097], dtype=torch.float64)
tensor([0.4516, 0.5484], dtype=torch.float64)
tensor([0.6312, 0.3688], dtype=torch.float64)
tensor([0.6200, 0.3800], dtype=torch.float64)
```

```
tensor([0.4100, 0.1800, 0.4100], dtype=torch.float64)
tensor([0.1095, 0.4259, 0.4646], dtype=torch.float64)
tensor([0.1600, 0.4100, 0.4300], dtype=torch.float64)
tensor([0.4400, 0.2500, 0.3100], dtype=torch.float64)
tensor([0.3597, 0.3003, 0.3400], dtype=torch.float64)
tensor([0.2700, 0.4602, 0.2698], dtype=torch.float64)
tensor([0.3997, 0.1600, 0.4403], dtype=torch.float64)
```

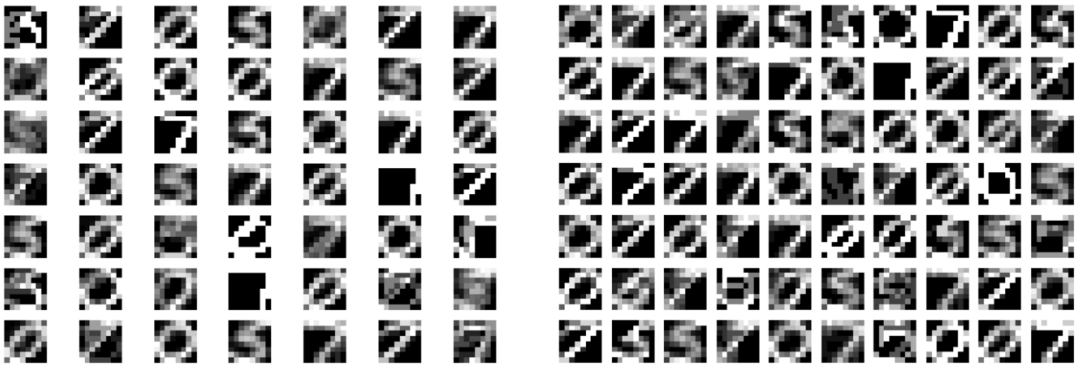
```
tensor([0.1600, 0.1899, 0.1800, 0.4701], dtype=torch.float64)
tensor([0.1600, 0.2700, 0.3800, 0.1900], dtype=torch.float64)
tensor([0.2500, 0.1800, 0.4400, 0.1300], dtype=torch.float64)
tensor([0.1500, 0.2020, 0.2380, 0.4100], dtype=torch.float64)
tensor([0.3099, 0.0800, 0.1800, 0.4301], dtype=torch.float64)
tensor([0.3998, 0.2892, 0.1810, 0.1300], dtype=torch.float64)
tensor([0.2890, 0.1800, 0.3200, 0.2110], dtype=torch.float64)
```



K = 2 K = 3 K = 4

```
tensor([0.0200, 0.0700, 0.2894, 0.1300, 0.2806, 0.0600, 0.1500],
       dtype=torch.float64)
tensor([0.1298, 0.0600, 0.0600, 0.2300, 0.1500, 0.2402, 0.1300],
       dtype=torch.float64)
tensor([0.2600, 0.0800, 0.0200, 0.1400, 0.2000, 0.0900, 0.2100],
       dtype=torch.float64)
tensor([0.1100, 0.1401, 0.2499, 0.1800, 0.2600, 0.0200, 0.0400],
       dtype=torch.float64)
tensor([0.1679, 0.2600, 0.1121, 0.0100, 0.2800, 0.1400, 0.0300],
       dtype=torch.float64)
tensor([0.0500, 0.0897, 0.1794, 0.0200, 0.2303, 0.0400, 0.3906],
       dtype=torch.float64)
tensor([0.2601, 0.0800, 0.1899, 0.1900, 0.1300, 0.0800, 0.0700],
       dtype=torch.float64)
```

```
tensor([0.0800, 0.1700, 0.1000, 0.1700, 0.0800, 0.0300, 0.0300, 0.0100, 0.2600,
        0.0700], dtype=torch.float64)
tensor([0.1600, 0.0900, 0.1898, 0.1002, 0.0400, 0.1600, 0.0200, 0.1000, 0.1000,
        0.0400], dtype=torch.float64)
tensor([0.0697, 0.0500, 0.0500, 0.0603, 0.0800, 0.1100, 0.2001, 0.1299, 0.1400,
        0.1100], dtype=torch.float64)
tensor([0.1405, 0.0200, 0.0700, 0.1200, 0.1495, 0.0400, 0.0900, 0.1100, 0.0100,
        0.2500], dtype=torch.float64)
tensor([0.1600, 0.0800, 0.1900, 0.0800, 0.1500, 0.0300, 0.0400, 0.0600, 0.1600,
        0.0500], dtype=torch.float64)
tensor([0.1099, 0.1394, 0.1000, 0.0200, 0.1100, 0.1207, 0.0400, 0.1300, 0.0600,
        0.1701], dtype=torch.float64)
tensor([0.0600, 0.0500, 0.1700, 0.1000, 0.1500, 0.1400, 0.0300, 0.0500, 0.2001,
        0.0500], dtype=torch.float64)
```



K = 7 K = 10

The code I ran is shown below

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.distributions as dist
import scipy as sp
from scipy import special
%matplotlib inline
```

In [4]:
```
Y = np.loadtxt('binarydigits.txt')
N, D = Y.shape
X = torch.tensor(Y)
```

In [5]:
```
"""
two functions that instantiate initial values for the parameters.
with equal probability for each Bernoulli model,
and the elements of the P matrix sampled uniformly from [0,1].
They both take as input 'k' the number of Bernoulli models
"""
def initial_pi(k):
    return torch.full((k,),1/k).double()

def initial_P(k,D):
    return torch.rand((k,D)).double()
```

In [6]:
```
"""
returns (N x K) matrix whose nth,kth element is the log-probability
of the nth image given the kth Bernoulli model
this is a done in a semi-vectorised approach,
"""
def prob_data_given_model(P,X):
    """big_X = torch.diag_embed(X).float()
    big_I_X = torch.diag_embed(torch.ones(X.size())-X).float()
    big_P = ((P.unsqueeze(0)).repeat(N,1,1)).float()
    big_I_P = (((torch.ones(P.size())-P).unsqueeze(0)).repeat(N,1,1)).float()
    prob_tensor = (torch.bmm(big_P,big_X) + torch.bmm(big_I_P,big_I_X)).double()"""

    big_X = torch.diag_embed(X)
    big_I_X = torch.diag_embed(torch.ones(X.size())-X)

    tensor = ((P@big_X[0]) + (torch.ones(P.size())-P)@big_I_X[0]).unsqueeze(0)

    for i in range(1,N):
        A = ((P@big_X[i]) + (torch.ones(P.size())-P)@big_I_X[i]).unsqueeze(0)
        tensor = torch.concat((tensor,A),0)


    prob_matrix = torch.prod(tensor,dim=2)
    return prob_matrix
```

In [7]:
```
"""
returns (N x K) matrix matrix whose nth,kth element is the joint probability
of the nth image and the kth Bernoulli model
"""
def prob_data(P,X,pi):
    return prob_data_given_model(P,X) @ torch.diag(pi)
```

In [8]:
```
"""
returns normalising factor required for the responsibilites
"""
def sum_over_models(P,X,pi):
    return torch.sum(prob_data(P,X,pi),1)
```

In [9]:
```
"""
returns matrix of responsibilities given the parameters P and pi
"""
def responsibility(P,X,pi):
    normalising = torch.div(torch.ones(sum_over_models(P,X,pi).size()),sum_over_mode
    normalising_mat = torch.diag(normalising)
    return normalising_mat @ prob_data(P,X,pi)
```

In [10]:
```
"""
returns the log-likelihood of the data given the parameters
"""
def log_likelihood(P,X,pi):
    big_X = torch.diag_embed(X).float()
    big_I_X = torch.diag_embed(torch.ones(X.size())-X).float()
    big_P = ((P.unsqueeze(0)).repeat(N,1,1)).float()
    big_I_P = (((torch.ones(P.size())-P).unsqueeze(0)).repeat(N,1,1)).float()
    prob_tensor = (torch.bmm(big_P,big_X) + torch.bmm(big_I_P,big_I_X)).double()
    prob_matrix = torch.prod(prob_tensor,dim=2)
    log_prob_single_image = torch.sum((prob_matrix @ torch.diag(pi)),dim=1)

    return torch.sum(torch.log(log_prob_single_image))
```

In [11]:
```
"""
updates the distribution of the latent variables (part of the M-step)
"""
def update_pi(R):
    return torch.sum(R,0)/N
```

In [12]:
```
"""
updates the benoulli parameter matrix (part of the M-step)
"""
def update_P(X,pi,R):
    numerator = torch.t(R) @ X
    #denominator = torch.diag(torch.div(torch.ones(pi.size()[0]),N*pi))
    denominator = torch.diag(torch.div(torch.ones(pi.size()[0]),torch.sum(R,0)))

    return torch.t(torch.t(numerator) @ denominator)
```

In [13]:
```
"""
Runs the EM-algorithm for N_iter iterations, and returns the losses
and learned parameters
"""
def EM_algorithm(K,X,N_iter):
    pi = initial_pi(K)
    P = initial_P(K,D)
    R = responsibility(P,X,pi)

    losses = []
    for _ in range(N_iter):
        R = responsibility(P,X,pi)
        likelihood = log_likelihood(P,X,pi)
        losses.append(likelihood.item())
```

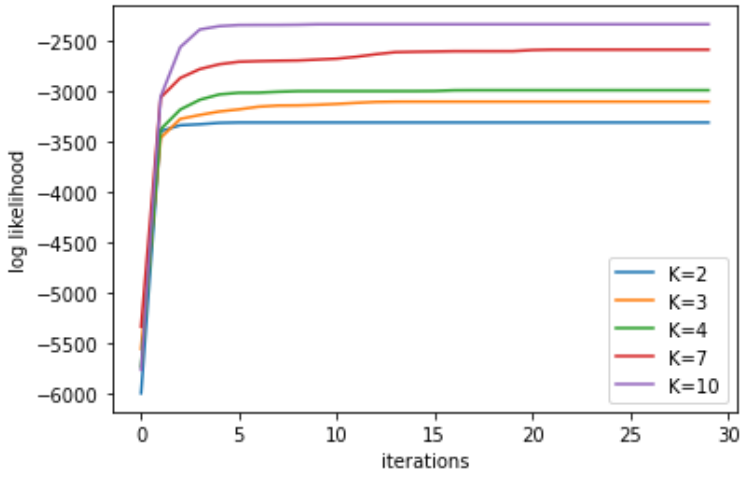
```
        pi = update_pi(R)
        P = update_P(X,pi,R)

    return losses, pi, P, R
```

In [34]:

```
K = [2,3,4,7,10]
labels = []
for k in K:
    losses, pi, P, R = EM_algorithm(k,X,30)
    plt.plot(np.arange(30),losses,label=f'K={k}')

plt.legend(loc="lower right")
plt.xlabel('iterations')
plt.ylabel('log likelihood')
plt.savefig('log_like_k.png')
plt.show()
```



In [35]:

```
def display_clusters(k):
    fig = plt.figure(figsize=(8,8))
    for i in range(7):
        losses, pi, P, R = EM_algorithm(k,X,30)
        P = P.numpy()
        for n in range(P.shape[0]):
            fig.add_subplot(10,P.shape[0],k*i+1+n)
            plt.imshow(np.reshape(P[n,:], (8,8)),
                       interpolation="None",
                       cmap='gray')
            plt.axis('off')
        print(pi)
    fig.subplots_adjust(wspace=0, hspace=0)
    plt.tight_layout()
    plt.show()
```

In [36]:

```
display_clusters(10)
```

```
tensor([0.0800, 0.1700, 0.1000, 0.1700, 0.0800, 0.0300, 0.0300, 0.0100, 0.2600,
        0.0700], dtype=torch.float64)
tensor([0.1600, 0.0900, 0.1898, 0.1002, 0.0400, 0.1600, 0.0200, 0.1000, 0.1000,
        0.0400], dtype=torch.float64)
tensor([0.0697, 0.0500, 0.0500, 0.0603, 0.0800, 0.1100, 0.2001, 0.1299, 0.1400,
        0.1100], dtype=torch.float64)
tensor([0.1405, 0.0200, 0.0700, 0.1200, 0.1495, 0.0400, 0.0900, 0.1100, 0.0100,
        0.2500], dtype=torch.float64)
tensor([0.1600, 0.0800, 0.1900, 0.0800, 0.1500, 0.0300, 0.0400, 0.0600, 0.1600,
```

Does the algorithm obtain the same solutions up to permutation:

For $K = 2$ the solutions are not the same up to permutation, because the data set consists of images of 3 types (0's, 5's and 7's), so for each run of the algorithm only probabilities corresponding to two image types can be found.

For $K = 3$ it seems that the same solutions are found the majority of the time. It looks like there are only small discrepancies between solutions corresponding to the same image (i.e the 7's don't all look identical). This makes sense as there are 3 image types, so the three Bernoulli models can correspond to each image type (1 to 1).

For $K > 3$ The algorithm does not obtain the same solutions up to permutation, as in this case there are more Bernoulli models than image types, so solutions corresponding to different image types can occur multiple times.

The cluster means are μ_k

$$\mu_k = \frac{\sum_{n=1}^{N} r_{nk}\mathbf{x}^{(n)}}{\sum_{n=1}^{N} r_{nk}}$$

This is equal to the kth row of the parameter matrix P, so in Figure (4) we are observing the cluster means. The majority of the time, good clusters are found, and the numbers 0, 5 and 7 can distinctly be made out. The model might be improved by introducing priors on P and π.

5. Decrypting Messages with MCMC

Part (a)

The transition statistics of letters and punctuation in English were learned on the large text provided in the question. The transition probability to go from symbol β to α was estimated using occurrences of pairs of symbols. The occurrences of symbols alone is not required, as we must normalise each column of the transition matrix anyway. Elements of the transition matrix P are given by:

$$\mathsf{P}_{\alpha\beta} = p(s_i = \alpha | s_{i-1} = \beta) \approx \frac{\text{Number of times } \beta \text{ is followed by } \alpha \text{ in reference text}}{Normaliser}$$

The normaliser is found by requiring each column to sum to 1. (a constant matrix was added before normalising to ensure ergodicity of the chain, this ensures the existence of a stationary distribution). The transition matrix is shown in visual form below. With the order of the columns/rows specified by the order of the symbols in 'symbols.txt' file.

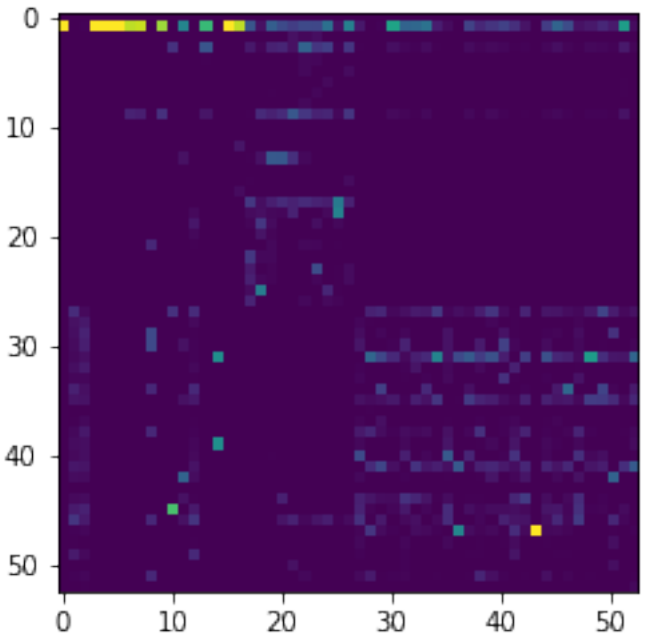


Figure 4: Visual representation of the transition matrix

The stationary distribution was found by multiplying the transition matrix by a randomly initialised vector 200 times (to ensure convergence). The stationary distribution is shown in the figure 5.

Part (b) and (c)

b). $\sigma(s)$ is the symbol that stands for the ~~encrypted text~~ symbol s in the encrypted text.

$\sigma^{-1}(s)$ is the symbol that stands for the symbol s in the message.

~~The latent variables for the symbols $\sigma(s)$~~

The latent variables $\sigma(s)$ for the different symbols s are NOT independent.
Because they depend on each other as specified by the same transition learned in part (a), but with its columns and rows permuted according to σ.

Let $e_1 e_2 \ldots e_n = e_{\{1:n\}}$ be an encryted English text.
Let $s_1 s_2 \ldots s_n = s_{\{1:n\}}$ be English text
given we know σ: $e_i = \sigma(s_i)$
$\sigma^{-1}(e_i) = s_i$

$$P(e_{\{1:n\}} \mid \sigma) = P(s_{\{1:n\}})$$

$$= P(s_1) \hat{\prod_{i=2}} P(s_i \mid s_{i-1})$$

$$= P(\sigma^{-1}(e_1)) \hat{\prod_{i=2}} P(\sigma^{-1}(e_i) \mid \sigma^{-1}(e_{i-1}))$$

where $P(\sigma^{-1}(e_1))$ is an element of the invariant distribution

c) We perform Metropolis-Hastings to sample the ~~space~~ space of permutations

Acceptance probability $a(\sigma, \sigma')$ is:

$$a(\sigma, \sigma') = \min\left(1, \frac{q(\sigma|\sigma')\, p(\sigma' | e_{\{1:n\}})}{q(\sigma'|\sigma)\, p(\sigma | e_{\{1:n\}})}\right)$$

The proposal is chosen by choosing two symbols s and s' at random, and swapping the corresponding encrypted symbols $\sigma(s)$ and $\sigma(s')$

Hence the probability to go from $\sigma \to \sigma'$ is equal to the probability to go from $\sigma' \to \sigma$

i.e ~~for~~ a symmetric proposal probability $\left(q(\sigma|\sigma') = q(\sigma'|\sigma)\right)$

Hence:

$$a(\sigma, \sigma') = \min\left(1, \frac{p(\sigma' | e_{\{1:n\}})}{p(\sigma | e_{\{1:n\}})}\right)$$

Assuming a uniform prior distribution over permutations, we use Bayes rule to get:

$$a(\sigma, \sigma') = \min\left(1, \frac{P(e_{\{1:n\}} | \sigma')}{P(e_{\{1:n\}} | \sigma)}\right)$$

Which can be calculated using the expression found in part (b).

Symbol	invariant probability	Symbol	invariant probability
=	2.438e-06	b	1.090e-02
	1.793e-01	c	1.951e-02
-	5.821e-04	d	3.751e-02
,	1.260e-02	e	9.940e-02
;	3.608e-04	f	1.746e-02
:	3.208e-04	g	1.622e-02
!	1.243e-03	h	5.314e-02
?	9.887e-04	i	5.426e-02
/	4.709e-06	j	8.124e-04
.	1.001e-02	k	6.357e-03
'	4.016e-06	l	3.018e-02
"	8.886e-06	m	1.948e-02
(	2.156e-04	n	5.773e-02
)	2.139e-04	o	6.031e-02
[	2.438e-06	p	1.446e-02
]	2.582e-06	q	7.408e-04
*	9.462e-05	r	4.707e-02
0	6.155e-05	s	4.957e-02
1	1.299e-04	t	7.103e-02
2	4.880e-05	u	2.041e-02
3	2.129e-05	v	8.040e-03
4	9.389e-06	w	1.849e-02
5	1.900e-05	x	1.399e-03
6	2.051e-05	y	1.437e-02
7	1.511e-05	z	5.609e-04
8	6.533e-05		
9	1.346e-05		
a	6.423e-02		

Figure 5: The stationary distribution over the symbols

Part (d)

The MH sampler was implemented and run on the encrypted text 10 times. In each case The chain was randomly initialised (i.e a random permutation was chosen to start), and the Markov chain converged to give a sensible message. Some convergences were faster than others, I found that in my implementations, the chain would always converge (given enough iterations of the algorithm), this does not necessarily mean that it will always converge in every case. I did not initialize my chain intelligently as it seemed to converge regardless of initialisation. Figure 6 shows the decryption of the first 60 symbols after every 100 iterations of the algorithm (in one case of random initalisation). The most iterations required for convergence in one case was 20,000 and the least was 5000. Probabilities involved in calculating

the acceptance probability were very small, so for numerical stability I worked with log-probs instead. The code I used to carry this out is included in the Appendix. The entire (semi)-decrypted message is also in the appendix. (for some reason figure 6 is also appearing in the appendix, I couldn't move it)

Part (e)

Some values in the transition matrix learned from the reference text, were 0, this affected the ergodicity of the chain, as I checked the matrix and found some of the diagonal elements to be 0. This means that the chain is not aperiodic and hence non-ergodic. I restored ergodicity by added a constant matrix to the transition matrix, all the elements of the added matrix were equal to the smallest element (excluding 0) in the (non-ergodic) transition matrix. This also makes the matrix irreducible (if it wasn't already) which is the other condition required for ergodicity.

Part (f)

answer begins on next page

e)i). Using symbol probabilities alone:

$$a(\sigma, \sigma') = \min\left(1, \frac{\prod_{i=1}^{n} P\left(\sigma^{-1}(e_i)\right)}{\prod_{i=1}^{n} P\left(\sigma^{-1}(e_i)\right)}\right)$$

This says nothing about the ordering of the letters, just gives information on which letters are most likely contained in the message.

(space)

i.e it would assign ' ' to the highest occurring symbol
then assign 'e' to the 2nd " " "
(∵ (space) is the most likely symbol as it's the highest occuring in reference text).
WILL NOT WORK

ii). Using a second order Markov chain:

$$P(s_{t+1:n}) = P(s_1)P(s_2|s_1)\prod_{i=3}^{n} P(s_i | s_{i-1}, s_{i-2})$$

Instead of 53×53 probabilities, we must store and compute 53×53×53 probabilities.

The conditions for existence of a stationary distribution are different for higher order Markov chains, so there may be not stationary distribution

iii. 2 symbols mapped to the same value.

The mapping σ is not 1 to 1, hence

$P(e_{[1:n]} | \sigma)$ cannot be calculated because $\sigma^{-1}(e_i)$ may be undefined

(WILL NOT WORK)

ii.

iv. It would work for Chinese, we just require a much larger reference text to learn transition probabilities.

7 Optimization

(answer begins on next page)

7. Optimization:

a). Extremize $f(x,y) = x+2y$ subject to $y^2+xy-1=0$

$(g(x,y)=0)$.

Formulate as constrained Lagrangian optimization problem.

$$\mathcal{L}(x,y,\lambda) = f(x,y) + \lambda g(x,y)$$
$$= x+2y+\lambda(y^2+xy-1)$$

$$\frac{\partial}{\partial x}\mathcal{L} = 1+\lambda y^* \quad \left.\right\} \text{set to } 0$$
$$\frac{\partial}{\partial x}\mathcal{L} = 2+2(2y^*+x^*) \quad \left.\right\}$$

$$\Rightarrow y^* = -\frac{1}{\lambda} \quad , \quad \lambda = \frac{-2}{2y^*+x^*}$$

① ②

① sub ②

$$\Rightarrow -2+x^*\lambda = -2$$

$$\lambda x^* = 0$$
$$x^* = 0 \quad (\lambda \neq 0 \text{ or } y^* \text{ undefined})$$

Using the constraint: $g(x,y)=0$

$$y^{*2}+x^*y^* = 1 \Rightarrow y^* = \pm 1$$

Extrema at: $(0,1)$ and $(0,-1)$

b) i. $f(x,a) = e^x - a$ →

$f(x,a) = 0$ is solved by $x = \ln(a)$

ii) Update equation: $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$

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

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

→

8 Eigenvalues as solutions of an optimization problem

(answer begins on next page)

8(a). $q_A(x) = x^T A x \qquad R_A(x) = \dfrac{x^T A x}{x^T x} \qquad x \in \mathbb{R}^n$

$$R_A(x) = \frac{x^T x \left(\dfrac{x^T}{(x^T x)^{1/2}} A \dfrac{x}{(x^T x)^{1/2}} \right)}{x^T x}$$

$$= \hat{x}^T A \hat{x}$$

$$= q_A(\hat{x})$$

[Where $\hat{x} = \dfrac{x}{(x^T x)^{1/2}}$ is the normalised x]

$\hat{x} \in \mathbb{R}^n$ and $\|\hat{x}\| = 1$

Hence $\sup_{x \in \mathbb{R}^n} \left[R_A(x) \right]$ is equivalent to

$$\sup_{x \in S} \left[q_A(x) \right]$$

Where $S = \{ x \in \mathbb{R}^n \mid \|x\| = 1 \}$

S is compact, therefore by the extreme value theorem of calculus,

$\sup_{x \in \mathbb{R}^n} R_A(x)$ is attained.

b). $x \in S$, $q_A(x) = x^T A x$

$$A = \sum_{i=1}^{n} \lambda_i \xi_i \xi_i$$

$$x = \sum_{i=1}^{n} a_i \xi_i \quad (\text{where } a_i = \xi_i^T x)$$

Therefore:

$$q_A(x) = \sum_{i,j,k} a_i a_k \xi_i^T \xi_j \xi_j^T \xi_k \lambda_j$$

$$= \sum_{i,j,k} \lambda_j a_i a_k \delta_{ij} \delta_{jk} \quad (\text{using orthonormality})$$

$$= \sum_{i=1}^{n} \lambda_i a_i^2$$

$$\leq \lambda_1 \sum_{i=1}^{n} a_i^2 \quad (\because \lambda_1 \geq \dots \geq \lambda_n)$$

using $\|x\| = 1$, $x^T x = 1$

$$\sum_{i=1}^{n} a_i^2 = 1$$

$\therefore$ $q_A(x) \leq \lambda_1 \sum_{i=1}^{n} a_i^2$

$$q_A(x) \leq \lambda_1$$

define $c \in \mathbb{R}_+^*$

if $x \in S$, then $cx \in \mathbb{R}^n$

using the results from (a),

$$q_A(x) \leq \lambda_1$$

$$\Rightarrow R_A(cx) \leq \lambda_1 \quad \forall c$$

as required

c) $x \in \mathbb{R}^n \setminus \text{span}\{\xi_1, \ldots, \xi_K\}$

$\Rightarrow \xi_i^T x = 0 \quad \forall i \in \{1, \ldots, K\}$ (using orthonormality)

$$a_i = 0 \quad \forall i \in \{1, \ldots, K\}$$

$$q_A(x) = \sum_{i=1}^{n} \lambda_i a_i^2$$

$$= \sum_{i=K+1}^{n} \lambda_i a_i^2$$

$$\leq \lambda_b \sum_{i=K+1}^{n} a_i^2 \quad \text{(let } \lambda_b \text{ be 2nd largest eigenvalue)}$$

$$= \lambda_b$$

$$< \lambda_1$$

$$q_A(x) < \lambda_1$$

$R_A(cx) < \lambda_1 \quad \forall c \in \mathbb{R}_+$ as required

Appendix

Code for question 5

In [546…

```
from collections import Counter
from itertools import groupby
from itertools import zip_longest

import random
import numpy as np
import matplotlib.pyplot as plt
import torch
import torch.distributions as dist
import scipy as sp
import math
from scipy import special
%matplotlib inline
```

In [547…

```
"""
The following defines all the texts as lists we can iterate over
"""

text_1 = list(open("englishtext.txt",encoding='utf8').read())
text_2 = [' ' if x=='\n' else x for x in text_1]
text = [x.lower() for x in text_2]

seen = text[0]
cleaned_text = [text[0]]
for i in text[1:]:
    if i == ' ':
        if i != seen:
            cleaned_text.append(i)
        seen = i
    else:
        cleaned_text.append(i)
        seen = i


symbols_1 = list(open("symbols.txt",encoding='utf8').read())
symbols = list(filter(('\n').__ne__, symbols_1))
symbol_set = set(symbols)
symbol_pairs = []
for i in symbols:
    for j in symbols:
        symbol_pairs.append((i,j))

symbol_pairs_set = set(symbol_pairs)
```

```
['=', ' ', '-', ',', ';', ':', '!', '?', '/', '.', "'", '"', '(', ')', '[', ']',
'*', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f',
'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
'x', 'y', 'z']
```

In [548…

```
message_1 = list(open('message.txt',encoding='utf8').read())
message = list(filter(('\n').__ne__, message_1))
```

In [590…

```
dictionary_1 = dict(Counter(cleaned_text))
```

In [550…

```
"""
cleaned_dict contains counts of all 53 symbols and no others
"""
cleaned_dict = dict((k, dictionary_1[k]) for k in set(dictionary_1) & symbol_set)
```

In [591…
```
"""
cleaned_dict_pairs contains counts of every consecutively occurring pairs of symbols
symbol_pairs is a list of all possible pair combinations (both orders included)
"""
dict_pairs = Counter(zip(cleaned_text, cleaned_text[1:]))
cleaned_dict_pairs = dict((k,dict_pairs[k]) for k in set(dict_pairs) & symbol_pairs_
```

In [592…
```
"""
define transition matrix full of zeros
resulting is matrix P where P[i][j] is proportional to prob from going from i to j (
"""

P = torch.zeros((53,53))

for i in range(53):
    for j in range(53):
        try:
            P[i][j] = cleaned_dict_pairs[(symbols[i],symbols[j])]
        except KeyError:
            pass

"""
defines a different matrix for test case
"""

test_P = (torch.ones_like(P) + P).t()
```

In [555…
```
"""
count_vector contains a vector of counts of the symbols (in the same order as P)
"""
count_vector = torch.zeros(53)
for i in range(53):
    count_vector[i] = cleaned_dict[symbols[i]]
```

In [594…
```
"""
we normalise the rows of P to sum to 1, then transpose to get the transition matrix
"""
normalisers = torch.sum(P,1)
normalised_P = (torch.diag(torch.div(torch.ones(normalisers.size()),normalisers)) @
```

In [595…
```
"""
the transition matrix estimate is not aperiodic (and may not be irreducible),
we make it aperiodic and irreducible by adding a*torch.ones(P.size()) to P.
where 'a' is decided by heuristics, we choose a=min(P)\0. we then normalise to make
the columns sum to one
"""
```

Out[595…
```
"\nthe transition matrix estimate is not aperiodic (and may not be irreducible),\nwe
make it aperiodic and irreducible by adding a*torch.ones(P.size()) to P.\nwhere 'a'
is decided by heuristics, we choose a=min(P)\x00. we then normalise to make\nthe col
umns sum to one\n"
```

In [596…
```
"""
finds the minimum value excluding 0 in the transition matrix
"""
minimum = 1
count= 0
```

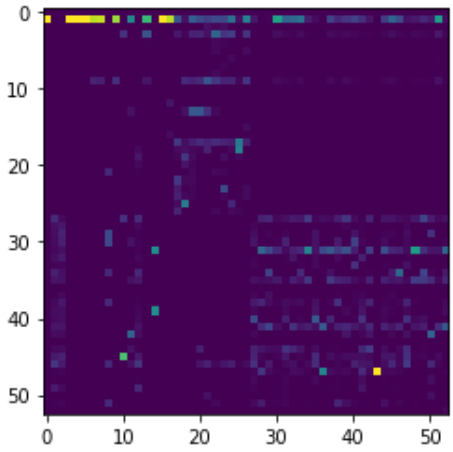
```
for i in normalised_P:
    for j in range(53):
        if i[j] != 0 and i[j]<minimum:
            minimum = i[j]
```

In [558…

```
regulariser = minimum*torch.ones_like(normalised_P)
irrep_P = normalised_P + regulariser

row_normalisers = torch.sum(irrep_P,0)
transition_M = irrep_P @ torch.diag(torch.div(torch.ones_like(row_normalisers),row_n

plt.imshow(transition_M, interpolation='nearest')
plt.savefig('transitionm.png')
plt.show()
```



In [597…

```
"""
we now find the stationary distribution
"""

init_p_1 = torch.full((53,),1/53)

for _ in range(100):
    init_p_1 = transition_M @ init_p_1
```

In [598…

```
stat_p = init_p_1
print(torch.sum(stat_p))
```

```
tensor(1.0000)
```

In [588…

```
"""
create a dictionary for the stationary distribution
"""

list_1 = [x.item() for x in stat_p]
invariant_dict = dict(zip(symbols,list_1))
```

In [490…

```
print ("{:<8} {:<15}".format('Symbol','invariant probability'))
for k,v in invariant_dict.items():
    print("{:<8} {:<15}".format(k, '{:.3e}'.format(v)))
```

```
Symbol   invariant probability
=        2.438e-06
```

```
         1.793e-01
-        5.821e-04
,        1.260e-02
;        3.608e-04
:        3.208e-04
!        1.243e-03
?        9.887e-04
/        4.709e-06
.        1.001e-02
'        4.016e-06
"        8.886e-06
(        2.156e-04
)        2.139e-04
[        2.438e-06
]        2.582e-06
*        9.462e-05
0        6.155e-05
1        1.299e-04
2        4.880e-05
3        2.129e-05
4        9.389e-06
5        1.900e-05
6        2.051e-05
7        1.511e-05
8        6.533e-05
9        1.346e-05
a        6.423e-02
b        1.090e-02
c        1.951e-02
d        3.751e-02
e        9.940e-02
f        1.746e-02
g        1.622e-02
h        5.314e-02
i        5.426e-02
j        8.124e-04
k        6.357e-03
l        3.018e-02
m        1.948e-02
n        5.773e-02
o        6.031e-02
p        1.446e-02
q        7.408e-04
r        4.707e-02
s        4.957e-02
t        7.103e-02
u        2.041e-02
v        8.040e-03
w        1.849e-02
x        1.399e-03
y        1.437e-02
z        5.609e-04
```

In [599…

```
"""
a dictionary specifying the order of the symbols in the transition matrix and
invariant distribution (key is symbol, key-value is index in matrix/vector)
"""

master_dict = dict(zip(symbols,range(0,53)))
```

In [600…

```
"""
dictionary corresponding to the identity permutation (for test cases)
"""
```

```
def identity_perm():
    identity_perm = dict(zip(symbols,symbols)), [symbols,symbols]
    return identity_perm
```

In [601…

```
"""
initialise random permutation
"""
def init_perm():
    L = [0,0]
    L_1 = symbols
    L_0 = random.sample(symbols,53)

    L[0] = L_0
    L[1] = L_1
    return dict(zip(L_0,L_1)), L
```

In [562…

```
"""
returns the invariant probability of the pre-image of a message symbol under
some permutation
"""
def invariant_prob_preimage(a,perm):
    index = master_dict[perm[a]]
    return stat_p[index]
```

In [563…

```
"""
returns the element of transition matrix indexed by the pre-images of of two message
symbols under some permutation
"""
def transition_prob_preimages(a,b,perm):
    index_1, index_2 = master_dict[perm[b]], master_dict[perm[a]]

    #return transition_M[index_1][index_2]
    return test_P[index_1][index_2]
```

In [564…

```
"""
finds the log-likelihood of the message given a permutation
"""
def log_likelihood(perm):
    """log_prob = torch.log(invariant_prob_preimage(message[0],perm))
    for i in range(len(message)-1):
        log_prob += torch.log(transition_prob_preimages(message[i],message[i+1],perm

    log_prob = 0
    for i in range(len(message)-1):
        log_prob += torch.log(transition_prob_preimages(message[i],message[i+1],perm

    return log_prob
```

In [576…

```
"""
swaps two random symbols to generate a new permutation
"""
def swap_values(u,v):
    a, b = random.randrange(53), random.randrange(53)
    while a == b:
        a, b = random.randrange(53), random.randrange(53)

    new_u = u.copy()
    new_v = v.copy()
```

```
    symbol_1 = v[0][a]
    symbol_2 = v[0][b]


    new_v[0][a] = symbol_2
    new_v[0][b] = symbol_1

    a = u[symbol_1]
    b = u[symbol_2]

    new_u[symbol_1] = b
    new_u[symbol_2] = a

    return new_u, new_v


"""test_u = identity_perm()[0]
test_v  = identity_perm()[1]
test_swap = swap_values(test_u,test_v)
print(test_u)
print(test_swap[0])
print(test_v)
print(test_swap[1])"""
```

Out[576…
```
'test_u = identity_perm()[0]\ntest_v  = identity_perm()[1]\ntest_swap = swap_values
(test_u,test_v)\nprint(test_u)\nprint(test_swap[0])\nprint(test_v)\nprint(test_swap
[1])'
```

In [573…
```
"""
print the message given some permutation
"""
def print_decrypted_message(perm,length):
    replacer = perm.get

    message_list = [replacer(n,n) for n in message]
    decrypted_message = ""
    for x in message_list:
        decrypted_message += x
    print(decrypted_message[:length])
```

In [569…
```
"""
returns boolean specifying whether to go from permutation t to permutation t+1
"""

def acceptance(perm_1,perm_2):
    a = log_likelihood(perm_2).item()
    b = log_likelihood(perm_1).item()
    """print(a)
    print(b)
    print(a-b)"""
    log_ratio = a - b
    """print(log_ratio)"""
    log_acceptance_prob = min(0,log_ratio)
    """print(log_acceptance_prob)"""
    a = dist.Uniform(0,1).sample()
    """print(math.log(a))"""
    if math.log(a) <= log_acceptance_prob:
        return True
    else:
        return False
```

In [570…
```
"""
defines one round of the MCMC sampler, starting from some starting permutation
"""
def MCMC_round(perm_dict,perm_list):
    new_dict, new_list = swap_values(perm_dict,perm_list)
    if acceptance(perm_dict,new_dict):
        return new_dict, new_list
    else:
        return perm_dict, perm_list
```

In [584…
```
def MCMC(N_iter):
    perm_dict, perm_list = init_perm()
    for i in range(N_iter):
        perm_dict, perm_list = MCMC_round(perm_dict,perm_list)
        if i%100 == 0:
            print_decrypted_message(perm_dict,60)
        if i == N_iter-1:
            print_decrypted_message(perm_dict,len(message))
```

In [587…
```
MCMC(20001)
```

```
*aq"sqsb=aercqwamq"bcrq =2arcw.2rqsrwc9q"sqtw8orcqew rq"rq9b
 akenknb=a!roklawkeborky=iaroldirknrloukenksl,trok!lyrkerkub
 akubkbnfa,roklawkunorkyfiaroldirkbrloekubkslgtrok,lyrkurken
 amubmbn-atromlacmunormp-darol,drmbrloemubmslgiromtlprmurmen
 miubibs-mtaoilmciusoaiw-pmaol,paibaloeiubinldraoitlwaiuaies
eaiubibs-af oilaciuso iy-pa ol,p ib lomiubinldr oifly iu ims
eaiubibs-af oilaciuso iy-ta olkt ib lomiubinldr oifly iu ims
laiuvivs-af oieaciuso iy-na oekn iv eoriuvitedm oifey iu irs
laiuvivsfa- oieaciuso iyfna oekn iv eoriuvitedm oi-ey iu irs
laiuvivsfat oieaciuso iyfna oekn iv eoriuvi-edm oitey iu irs
oaiuvivlfat sieaciuls iyfna se-n iv esriuvikedm sitey iu irl
oaitkiklfac sieamitls iyfra se-r ik esnitkivedu sicey it inl
oaidkiklfac rieamidlr iyfsa re-s ik ernidkivetu ricey id inl
oaidhihlfac rieamidlr iyfsa re-s ih ernidhivetu ricey id inl
oaidhihlfac rieamidlr iyfsa re-s ih ernidhivetu ricey id inl
oaidhihlfac rieamidlr iyfsa regs ih ernidhivetu ricey id inl
oaidhihlfac rieamidlr iyfsa regs ih ernidhivetu ricey id inl
oaidhihufac rieamidur iyfsa regs ih ernidhivetl ricey id inu
oaidhihufac rieamidur iyfsa regs ih ernidhivetl ricey id inu
oaidhihufac rieamidur iyfsa regs ih ernidhivetl ricey id inu
oaidhihufac rieamidur iyfsa regs ih ernidhivetl ricey id inu
oaidhihufac rieamidur iyfsa regs ih ernidhivetl ricey id inu
oaidlilufac rieamidur iyfna regn il ersidliveth ricey id isu
oaidlilufaw rieamidur iyfna revn il ersidligeth riwey id isu
oaidlilufaw rieamidur iyfna revn il ersidligeth riwey id isu
oaidlilufaw rieamidur iyfna revn il ersidligeth riwey id isu
oaidlilufaw rieamidur iyfna revn il ersidligeth riwey id isu
oaidlilufaw rieamidur iyfna revn il ersidligeth riwey id isu
oaidlilunaw rieamidur iynfa revf il ersidligeth riwey id isu
eaidlilunaw rioamidur iynfa rovf il orsidlikoth riwoy id isu
eaidlilunaw rioamidur iynfa rovf il orsidlikoth riwoy id isu
eaidlilunaw rioamidur iynfa rovf il orsidlikoth riwoy id isu
eaidlilunaw rioamidur iynfa rovf il orsidlikoth riwoy id isu
eaidlilunaw rioamidur iynfa rovf il orsidlikoth riwoy id isu
eaidlilunaw rioamidur iynfa rovf il orsidlikoth riwoy id isu
eaidlilunaw rioamidur iynfa rovf il orsidlikoth riwoy id isu
eaidlilunaw rioavidur iynfa romf il orsidlikoth riwoy id isu
eaidlilunaw rioavidur iynfa romf il orsidlikoth riwoy id isu
enidliluanw rionvidur iyafn romf il orsidliboth riwoy id isu
enidlilounw rianvidor iyufn ramf il arsidlibath riway id iso
```

```
enidlilounw rianvidor iyufn ramf il arsidlibath riway id iso
inedlelounw reanvedor eyufn ramf el arsedlebath reway ed eso
inedleloung reanvedor eyumn rafm el arsedlebath regay ed eso
inedleloung reanvedor eyumn rafm el arsedlebath regay ed eso
inedleloung reanvedor eyumn rafm el arsedlebath regay ed eso
inedleloung reanvedor eyumn rafm el arsedlebath regay ed eso
inedyeyoung reanvedor elumn rafm ey arsedyebath regal ed eso
inedyeyoung reanvedor elumn rafm ey arsedyebath regal ed eso
inedyeyoung reanvedor elumn rafm ey arsedyebath regal ed eso
inedyeyoung reanvedor emuln rafl ey arsedyebath regam ed eso
inedyeyoung reanvedor emuln rafl ey arsedyebath regam ed eso
inedyeyoung reanvedor emuln rafl ey arsedyebath regam ed eso
inedyeyoung reanvedor emuln rafl ey arsedyebath regam ed eso
inedyeyoung reanvedor emuln rafl ey arsedyebath regam ed eso
inedyeyoung reanvedor emuln rafl ey arsedyebath regam ed eso
inedyeyoung reanvedor emuln rafl ey arsedyebath regam ed eso
inedyeyoung reanvedor emuln rafl ey arsedyebath regam ed eso
inedyeyoung reanvedor efuln raml ey arsedyebath regaf ed eso
inedyeyoung reanvedor efuln raml ey arsedyebath regaf ed eso
in dy younger anv dore fulneramle years dy bather gafe de so
in dy younger anf dore vulnerable years dy mather gave de so
in dy younger anf dore vulnerable years dy mather gave de so
in dy younger anf dore vulnerable years dy mather gave de so
in dy younger anm dore vulnerable years dy father gave de so
in dy younger anm dore vulnerable years dy father gave de so
in dy younger anm dore vulnerable years dy father gave de so
in dy younger anm dore vulnerable years dy father gave de so
in dy younger anm dore vulnerable years dy father gave de so
in dy younger anm dore vulnerable years dy father gave de so
in dy younger anm dore vulnerable years dy father gave de so
in dy younger anm dore vulnerable years dy father gave de so
in dy younger anm dore vulnerable years dy father gave de so
in dy younger anm dore vulnerable years dy father gave de so
in dy younger anm dore vulnerable years dy father gave de so
in dy younger anm dore vulnerable years dy father gave de so
in dy younger anm dore vulnerable years dy father gave de so
in dy younger anm dore vulnerable years dy father gave de so
in dy younger anm dore vulnerable years dy father gave de so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
```

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

```
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me some advice that ixve been
turning over in my mind ever since. *whenever you feel like criticizing any one,* he
told me, *just remember that all the people in this world havenxt had the advantages
that youxve had.* he didnxt say any more but wexve always been unusually communicati
ve in a reserved way, and i understood that he meant a great deal more than that. in
consequence ixm inclined to reserve all judgments, a habit that has opened up many c
urious natures to me and also made me the victim of not a few veteran bores. the abn
ormal mind is quick to detect and attach itself to this quality when it appears in a
normal person, and so it came about that in college i was unjustly accused of being
a politician, because i was privy to the secret griefs of wild, unknown men. most of
the confidences were unsought--frequently i have feigned sleep, preoccupation, or a
hostile levity when i realized by some unmistakable sign that an intimate revelation
was quivering on the horizon--for the intimate revelations of young men or at least
the terms in which they e?press them are usually plagiaristic and marred by obvious
suppressions. reserving judgments is a matter of infinite hope. i am still a little
afraid of missing something if i forget that, as my father snobbishly suggested, and
i snobbishly repeat a sense of the fundamental decencies is parcelled out unequally
at birth.
```

```
In [ ]:
```

```
h)*3f*fk6)n5x*o)]*3kx5*!6i)5xobi5*f5ox *3f*'od15x*no!5*35* k
e mr*m*k( psnmo tmrknsma(i snobism*sonwmr*m?olysnmpoasmrsmwk
a urfufkx psnue turknsuoxi snebisufsenwurfumelysnupeosursuwk
a uwfufip xsnue tuwinsuopk snebksufsenluwfumerysnuxeosuwsuli
n egfefip xoteu aegitoespk otubkoefoutlegfemuryotexusoegoeli
n egdedip xoteu segitoeapk otuvkoedoutlegdemuryotexuaoegoeli
n egdedis xoteu pegitoeash otuvhoedoutlegdemuryotexuaoegoeli
n ecyeyis xoleu meciloeksh oluvhoeyoultecyepurdolexukoecoeti
n eckekih xoleu meciloeyhs oluvsoekoulteckepurdolexuyoecoeti
n eckekih poleu meciloeyhs oluvsoekoulteckegurdolepuyoecoeti
ne ck kuhepol aem culo yhseolavso koalt ck gardol payo co tu
ne ck kuhefol aem culo yhseolavso koalt ck gardol fayo co tu
en ck kuhnfol anm culo yhsnolavso koalt ck gardol fayo co tu
en ck kuhniol anm culo yhsnolavso koalt ck gardol iayo co tu
en ck kuhniol anm culo dhsnolavso koalt ck garyol iado co tu
en ck kuhnior anm curo fhsnoravso koart ck galyor iafo co tu
on ck kuhnier anm cure fhsneravse keart ck galyer iafe ce tu
on ck kihnuer anm cire fhsneravse keart ck galyer uafe ce ti
on ck kihnuer anm cire fhsneravse keart ck galyer uafe ce ti
on ck kihnuer anm cire bhsnerapse keart ck galyer uabe ce ti
on ck kihnuer anm cire fhsnerapse keart ck galyer uafe ce ti
on ck kihnuer anm cire fhsnerapse keart ck galyer uafe ce ti
on ck kihnuer anm cire fhsnerapse keart ck galyer uafe ce ti
on ck kihnuer anm cire fhsnerapse keart ck galyer uafe ce ti
on ck kihnuer anm cire fhsnerapse keart ck galyer uafe ce ti
or ck kihruen arm cine fhsrenapse keant ck galyen uafe ce ti
or ck kihruen arm cine fhsrenapse keant ck galyen uafe ce ti
or ck kihruen arm cine fhlrenaple keant ck gasben uafe ce ti
or ck kihruen arm cine fhlrenaple keant ck gasben uafe ce ti
or cy yihruen arm cine fhlrenaple yeant cy gasben uafe ce ti
or cy yihruen ard cine fhlrenaple yeant cy gasben uafe ce ti
or cy yihruen ard cine fhlrenaple yeant cy gasben uafe ce ti
or cy yipruen ard cine vplrenafle yeant cy gasben uave ce ti
or cy yipruen ard cine vplrenafle yeant cy gasben uave ce ti


or cy yipruen ard cine vplrenafle yeant cy gasben uave ce ti
ir cy yopruen ard cone vplrenafle yeant cy gasben uave ce to
ir cy yopruen ard cone vplrenafle yeant cy gasben uave ce to
in by yopnuer and bore vplnerafle yeart by gascer uave be to
in gy yopnuer and gore vplnerable yeart gy fascer uave ges to
in gy younper and gore vulnerable years gy fatcer pave ge so
in gy younper and gore vulnerable years gy fatcer pave ge so
in fy younper and fore vulnerable years fy gatcer pave fe so
in fy younker and fore vulnerable years fy gatcer kave fe so
in fy younker and fore vulnerable years fy gatcer kave fe so
in fy younker and fore vulnerable years fy gatcer kave fe so
in fy younker and fore vulnerable years fy gatcer kave fe so
in fy younker and fore vulnerable years fy gatcer kave fe so
in hy younker and hore vulnerable years hy gatwer kave he so
in hy younker and hore vulnerable years hy gatwer kave he so
in hy younker and hore vulnerable years hy gatwer kave he so
in hy younker and hore vulnerable years hy gatwer kave he so
in wy younker and wore vulnerable years wy father kave we so
in cy younker and core vulnerable years cy father kave ce so
in cy younker and core vulnerable years cy wather kave ce so
in cy younker and core vulnerable years cy wather kave ce so
in cy younker and core vulnerable years cy wather kave ce so
in cy younker and core vulnerable years cy wather kave ce so
in cy younker and core vulnerable years cy wather kave ce so
in cy younker and core vulnerable years cy wather kave ce so
in cy younfer and core vulnerable years cy wather fave ce so
in wy youncer and wore vulnerable years wy father cave we so
in wy youncer and wore vulnerable years wy father cave we so
in wy youncer and wore vulnerable years wy father cave we so
in wy youncer and wore vulnerable years wy father cave we so
in wy youncer and wore vulnerable years wy father cave we so
in cy younger and core vulnerable years cy father gave ce so
in cy younger and core vulnerable years cy father gave ce so
in my younger and more vulnerable years my father gave me so
in my younger and more vulnerable years my father gave me so
```

Figure 6: The first 60 symbols of the decrypted message after every 100 iterations of the algorithm. The initialisation of the parameter matrix was random

In [2]:
```
Y = np.loadtxt('binarydigits.txt')
N, D = Y.shape
```

In [3]:
```
"""
relative log-probability of model (a), bernoulli distribution with all p_d = 0.5
"""
log_prob_M_a = -N*D*np.log(2)
print(log_prob_M_a)
```

```
-4436.14195558365
```

In [4]:
```
"""
relative log-probability of model (b)
"""
log_beta_alphabeta = sp.special.betaln(alpha,beta)
alpha_b = 1 + np.sum(np.sum(Y,axis=0))
beta_b = 1 + D*N - np.sum(np.sum(Y,axis=0))

log_prob_M_b = sp.special.betaln(alpha_b,beta_b)
print(log_prob_M_b)
```

```
-4283.721342577359
```

In [6]:
```
"""
relative log-prob of model (c)
"""
x_sum = np.sum(Y,axis=0)
vec_alpha_c = 1 + x_sum
vec_beta_c = 1 + N - x_sum

log_beta_vec = sp.special.betaln(vec_alpha_c,vec_beta_c)

log_prob_M_c = np.sum(log_beta_vec)
print(log_prob_M_c)
```

```
-3851.1957439211315
```

In [14]:
```
relative_probs = [np.exp(4300+log_prob_M_a),np.exp(4300+log_prob_M_b),np.exp(4300+log_prob_M_c)]
print(relative_probs)
```

```
[7.486863833253369e-60, 11741703.013759049, 8.188641720544932e+194]
```

Figure 7: Code for question 2