Summative Assignments

COMP0086

Candidate number : GKLN9

Contents

1 Models for Binary Vectors

1.1

A model for this data would be a model composed of D random variables that are binary with values 0 or 1, not a model like normal distributions that assume continuous real-valued data. Plus, binary data cannot be normally distributed as its data cannot be spatially clustered around a mean so a normal distribution would be a poor estimator.

1.2

We assume that observations in $\mathcal{D} = \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(N)}\}$ were i.i.d sampled from $P(\boldsymbol{x} \mid \mathbf{p})$

$$\begin{aligned}\hat{\mathbf{p}}_{ML} &= \operatorname*{argmax}_p \prod_{i=1}^{N} P(\boldsymbol{x}^{(i)} \mid \mathbf{p}) \\ &= \operatorname*{argmax}_p \prod_{i=1}^{N} \prod_{d=1}^{D} p_d^{x_d^{(i)}} (1-p_d)^{(1-x_d^{(i)})}\end{aligned}$$

Taking the log-likelihood we get:

$$\hat{\mathbf{p}}_{ML} = \arg\max_p \sum_{i=1}^{N} \sum_{d=1}^{D} x_d^{(i)} \log(p_d) + (1 - x_d^{(i)}) \log(1-p_d)$$

Thus $\hat{p}_{ML}$ is the solution of,

$$\nabla_{\mathbf{p}} \sum_{i=1}^{N} \sum_{d=1}^{D} x_d^{(i)} \log(p_d) + \left(1 - x_d^{(i)}\right) \log(1-p_d) = 0$$

which is satisfied by $\forall j \in [D]$ by

$$\frac{\partial}{\partial p_j} \sum_{i=1}^{N} \sum_{d=1}^{D} x_d^{(i)} \log(p_d) + (1 - x_d^{(i)}) \log(1-p_d) = 0$$

$$\Leftrightarrow \sum_{i=1}^{N} \frac{\partial}{\partial p_j} \left(x_d^{(i)} \log(p_d) + (1 - x_d^{(i)}) \log(1-p_d) \right) = 0$$

$$\Leftrightarrow \sum_{i=1}^{N} \frac{x_j^{(i)}}{p_j} - \frac{\left(1 - x_j^{(i)}\right)}{1 - p_j} = 0 \quad (1)$$

$$\Leftrightarrow \frac{1}{p_j} \sum_{i=1}^{N} x_j^{(i)} = \frac{1}{1 - p_j} \sum_{i=1}^{N} 1 - x_j^{(i)} = \frac{N}{1 - p_j} - \frac{1}{1 - p_j} \sum_{i=1}^{N} x_j^{(i)}$$

$$\Leftrightarrow \left(\sum_{i=1}^{N} x_j^{(i)} \right) \left(\frac{1}{p_j} + \frac{1}{1 - p_j} \right) = \frac{N}{1 - p_j}$$

$$\Leftrightarrow \left(\sum_{i=1}^{N} x_j^{(i)} \right) \left(\frac{1}{p_j \left(1 - p_j\right)} \right) = \frac{N}{1 - p_j}$$

$$\Leftrightarrow \sum_{i=1}^{N} x_j^{(i)} = \frac{N p_j \left(1 - p_j\right)}{\left(1 - p_j\right)}$$

$$\Rightarrow \hat{p}_j = \frac{1}{N} \sum_{i=1}^{N} x_j^{(i)}$$

$$\Rightarrow \hat{\mathbf{p}}_{ML} = \left(\frac{1}{N} \sum_{i=1}^{N} x_1^{(i)}, \ldots, \frac{1}{N} \sum_{i=1}^{N} {x_D}^{(i)} \right)$$

1.3

By Bayes' rule:

$$P(\mathbf{p} \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \mathbf{p}) P(\mathbf{p})}{P(\mathcal{D})} \propto P(\mathcal{D} \mid \mathbf{p}) P(\mathbf{p})$$

$$\Rightarrow \hat{\mathbf{p}}_{\text{MAP}} = \underset{\mathbf{p}}{\text{argmax}} P(\mathbf{p} \mid \mathcal{D}) = \underset{p}{\text{argmax}} \log P(\mathbf{p} \mid \mathcal{D})$$

$$= \underset{\mathbf{p}}{\text{argmax}} [\log P(\mathcal{D} \mid \mathbf{p}) + \log P(\mathbf{p}) - \log P(\mathcal{D})]$$

$-\log P(\mathcal{D})$ is constant with respect to $\mathbf{p}$, so we discard it (as it vanished when differentiating with respect to $\mathbf{p}$), giving:

$$\hat{\mathbf{p}}_{\text{MAP}} = \underset{p}{\text{argmax}}[\log P(\mathcal{D} \mid \mathbf{p}) + \log P(\mathbf{p})]$$

As beta priors are independent, then,

$$\log P(\mathbf{p}) = \sum_{d=1}^{D} \log \frac{1}{\beta(\alpha, \beta)} + (\alpha - 1)\log(p_d) + (\beta - 1)\log(1 - p_d)$$

$$\Rightarrow \hat{\mathbf{p}}_{\text{MAP}} = \underset{\mathbf{p}}{\text{argmax}} \left[\sum_{i=1}^{N} \sum_{d=1}^{D} x_d^{(i)} \log(p_d) + \left(1 - x_d^{(i)}\right) \log(1 - p_d) + \sum_{d=1}^{D} (\alpha - 1)\log(p_d) + (\beta - 1)\log(1 - p_d) \right]$$

We disregard the Beta constant as it differentiates to 0 w.r.t p_j. Analogously for the ML estimate, $\hat{\mathbf{p}}_{MAP}$ solves the equation:

$$\sum_{i=1}^{N} \sum_{d=1}^{D} \frac{\partial}{\partial p_j} (x_d^{(i)} \log(p_d) + \left(1 - x_d^{(i)}\right) \log(1 - p_d)) + \sum_{d=1}^{D} \frac{\partial}{\partial p_j} ((\alpha - 1)\log(p_d) + (\beta - 1)\log(1 - p_d)) = 0 \quad (2)$$

for all $j \in [D]$

The term inside the first sum is given by (1), and:

$$\sum_{d=1}^{D} \frac{\partial}{\partial p_j} ((\alpha - 1)\log(p_d) + (\beta - 1)\log(1 - p_d)) = \frac{(\alpha - 1)}{p_j} - \frac{(\beta - 1)}{1 - p_j}$$

Thus, (2) is equivalent to,

$$\left[\sum_{i=1}^{N} \frac{x_j^{(i)}}{p_j} - \frac{\left(1 - x_j^{(i)}\right)}{1 - p_j} \right] + \frac{(\alpha - 1)}{p_j} - \frac{(\beta - 1)}{1 - p_j} = 0$$

$$\Leftrightarrow \left(\sum_{i=1}^{N} \frac{x_j^{(i)}}{p_j} \right) - \frac{N}{1 - p_j} + \frac{1}{1 - p_j} \left(\sum_{i=1}^{N} x_j^{(i)} \right) + \frac{(\alpha - 1)}{p_j} - \frac{(\beta - 1)}{1 - p_j} = 0$$

$$\Leftrightarrow \left(\frac{1}{p_j - 1}\right)\left(\sum_{i=1}^{N} x_j^{(i)}\right) + \left(\sum_{i=1}^{N} x_j^{(i)}\right) + \left(\frac{1}{p_j - 1}\right)(\alpha - 1) - (\beta - 1) = N$$

$$\Leftrightarrow \frac{1}{p_j}\left(\sum_{i=1}^{N} x_j^{(i)}\right) + \frac{1}{p_j}(\alpha - 1) = N + \alpha + \beta - 2$$

$$\Rightarrow (\hat{p}_{MAP})_j = \frac{S_j + (\alpha - 1)}{N + \alpha + \beta - 2}, \forall j \in [D]$$

Where $S_j = \sum_{i=1}^{N} x_j^{(i)}$, for $j \in [D]$.

$$\Rightarrow \hat{\mathbf{p}}_{\text{MAP}} = \left(\frac{S_1 + (\alpha - 1)}{N + \alpha + \beta - 2}, \ldots, \frac{S_D + (\alpha - 1)}{N + \alpha + \beta - 2}\right)$$

1.4

Code to learn ML estimates:

```
#As we know from question b) parameter of jth pixel given by ML is
    the empirical mean of binary values of the jth pixel in
#the N images (samples)

# ML estimate for each pixel
Bparams = np.mean(Y, axis=0)

# Reshape the parameters into an 8x8 image
image = np.reshape(Bparams, (8, 8))

# Display the learned parameter vector as an image
plt.figure()
plt.imshow(image, interpolation="None", cmap='gray')
plt.title("Learned Parameter Image")
plt.axis('off')
plt.show()
```

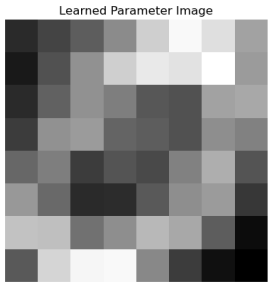

Figure 1: Learned parameter image

1.5

Code to learn MAP parameters:

```
# Calculate the MAP estimates for each pixel
# Were going to use our results found in c) where alpha=beta=3

N=100
alpha = 3
beta = 3
map_params = (np.sum(Y, axis=0) + alpha - 1) / (N + alpha + beta -
    2)

# Reshape the MAP parameters into an 8x8 image
map_image = np.reshape(map_params, (8, 8))

# Display the learned parameter vector as an image
plt.figure()
plt.imshow(map_image, interpolation="None", cmap='gray')
plt.title("MAP Parameter Image (alpha=3, beta=3)")
plt.axis('off')
plt.show()
```

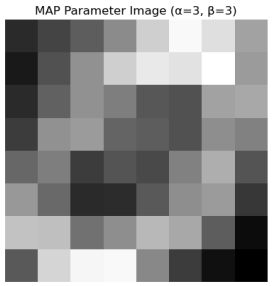

Figure 2: MAP estimates as an image

Note: using greyscale, there is no clear visual difference between the ML and MAP estimate. In our context,

$$\hat{\mathbf{p}}_{\text{MAP}} = \frac{100\hat{\mathbf{p}}_{\text{ML}} + 2}{104}$$

Which is too small of a difference to be captured visually using greyscale here.

In our scenario, we employ a symmetric beta distribution as the prior, which avoids bias towards extreme pixel values. MAP estimation is particularly advantageous when we have reliable prior knowledge about the distribution's parameters, making it a more suitable choice for smaller datasets. This preference stems from the fact that priors are independent of the dataset size. In cases where the dataset is large, the likelihood term generally overshadows the prior, guiding the estimation. Incorporating prior knowledge into our model is akin to adding synthetic observations; however, these don't always align with the true parameter distribution. If the prior knowledge is inaccurate, integrating it might lead to poorer outcomes than relying solely on likelihood. Conversely, precise prior knowledge can steer the model closer to the true parameter estimates. For smaller datasets, a symmetric prior acts as a regularizer, biasing parameter estimates towards 0.5, preventing pixel values from being too close to 0 or 1, which could indicate overfitting. In contrast, for larger datasets, ML becomes preferable as the likelihood provides more accurate results, and the addition of a prior would merely increase computational load without significantly influencing the outcome.

In our binary data problem, the dataset of 100 images can be considered moderately sized. It's not overly large, where the influence of the prior might be negligible, nor is it too small where data scarcity could lead to significant overfitting. Thus a symmetric beta prior in the MAP estimation is likely a good choice. It provides the benefits of regularization and balance, without introducing undue bias, making it well-suited for this moderately sized dataset.

2 Model Selection

Let a, b, and c, be our three models each with an equal prior probability, based on a dataset of N binary images with D pixels each.

To calculate the relative probability of each of these models we need to evaluate the likelihood of the observed data under each of them, i.e $P(\mathcal{D}|\mathcal{M})$. Where,

$$P(\mathcal{D}|\mathcal{M}) = \int_0^1 P(\mathcal{D}|\mathcal{M}, \theta) P(\theta|\mathcal{M}) d\theta$$

1. Model a: All D components (pixels) are generated from a Bernoulli distribution with $p_d = 0.5$
2. Model b: All D components are generated from Bernoulli distributions with unknown but identical p_d
3. Model c: Each of the D components is Bernoulli distributed with separate, unknown p_d.

The prior probability for each model is $\frac{1}{3}$. Thus, the posterior probability for each model will be :

$$P(\mathcal{M}|\mathcal{D}) = \frac{L(\mathcal{M}) \times \frac{1}{3}}{P(\mathcal{D})}$$

Where evidence is given by : $P(\mathcal{D}) = L(a) \times \frac{1}{3} + L(b) \times \frac{1}{3} + L(c) \times \frac{1}{3}$

2.1 Model *a*

Likelihood:

$$L(a) = \prod_{i=1}^{N} P(\boldsymbol{x}^{(i)} \mid \mathbf{p}) = \prod_{i=1}^{N} \prod_{d=1}^{D} p_d^{x_d^{(i)}} (1 - p_d)^{(1 - x_d^{(i)})}$$

Since $p_d = 0.5$ then this simplifies to :

$$L(a) = \prod_{i=1}^{N} \prod_{d=1}^{D} \frac{1}{2^{x_d^{(i)}}} \frac{1}{2^{(1-x_d^{(i)})}} = \prod_{i=1}^{N} \prod_{d=1}^{D} \frac{1}{2}$$

$$\Rightarrow L(a) = (0.5^D)^N = 0.5^{DN}$$

2.2 Model *b*

Likelihood (assuming uniform prior for p):

$$L(b) = P(\mathcal{D}|b) = \int P(\mathcal{D}|b, \mathbf{p}_b) \times 1 d\mathbf{p}_b$$

$$\Rightarrow L(b) = \int_0^1 \prod_{i=1}^{N} \prod_{d=1}^{D} p^{x_d^{(i)}} (1-p)^{(1-x_d^{(i)})} dp$$

$$= \prod_{i=1}^{N} \prod_{d=1}^{D} \int_0^1 p^{x_d^{(i)}} (1-p)^{(1-x_d^{(i)})} dp$$

$$= \prod_{i=1}^{N} \int_0^1 p^{s_i} (1-p)^{D-s_i} dp$$

where s_i is the sum of $\boldsymbol{x}^i$, i.e the number of 1's in $\boldsymbol{x}^i$

$$\Rightarrow L(b) = \int_0^1 p^S (1-p)^{ND-S} dp$$

Where S is the total number of successes (1s) in the whole dataset $\mathcal{D}$. Observe that this integral is equal to the beta function.

$$\Rightarrow L(b) = B(S+1, ND-S+1)$$

2.3 Model c

Likelihood (assuming uniform priors for each p_d):

$$L(c) = \int P(\mathcal{D}|c, \mathbf{p}_c) d\mathbf{p}_c$$

$$= \int_0^1 \dots \int_0^1 \prod_{i=1}^{N} P(\boldsymbol{x}^{(i)} \mid \mathbf{p})\, dp_1 \dots dp_D$$

$$= \int_0^1 \dots \int_0^1 \prod_{i=1}^{N} \prod_{d=1}^{D} p_d^{x_d^{(i)}} (1-p_d)^{(1-x_d^{(i)})}\, dp_1 \dots dp_D$$

$$L(c) = \int_0^1 \dots \int_0^1 \prod_{d=1}^{D} p_d^{S_d} (1-p_d)^{N-S_d}\, dp_1 \dots dp_D$$

where S_d is the number of successes for the d-th component in $\mathcal{D}$. Given that each pixel is independent, this can be simplified to a product of integrals :

$$\Rightarrow L(c) = \prod_{d=1}^{D} \int_0^1 p_d^{S_d}(1-p_d)^{N-S_d}\, dp_d = \prod_{d=1}^{D} B(S_d+1, N-S_d+1)$$

2.4 Posterior probabilities of models

Code is :

```
import numpy as np
import scipy.special as sc

# Load the data set
file_path = r'C:\Users\chdor\OneDrive\Desktop\binarydigits.txt'
Y = np.loadtxt(file_path)

# Prior probability for each model
prior_prob = 1/3

# Model (a)
log_unnormalized_post_a = np.log(prior_prob) - 64 * 100 * np.log(2)

# Model (b)
log_unnormalized_post_b = np.log(prior_prob) + sc.betaln(np.sum(Y)
    + 1, 100*64 - np.sum(Y) + 1)

# Model (c)
log_unnormalized_post_c = np.log(prior_prob) + np.sum([sc.betaln(np
    .sum(Y, axis=0)[i] + 1, 100 - np.sum(Y, axis=0)[i] + 1) for i
    in range(64)])

# Use logsumexp to calculate the log of the sum of exponentials (
    evidence)
log_evidence = sc.logsumexp([log_unnormalized_post_a,
    log_unnormalized_post_b, log_unnormalized_post_c])

# Normalize the log probabilities to get the log posterior
    probabilities
log_post_a = log_unnormalized_post_a - log_evidence
log_post_b = log_unnormalized_post_b - log_evidence
log_post_c = log_unnormalized_post_c - log_evidence

# Convert log posterior probabilities to actual probabilities
post_a = np.exp(log_post_a)
post_b = np.exp(log_post_b)
post_c = np.exp(log_post_c)

# Print the posterior probabilities for each model
print("Posterior Probability for Model (a):", post_a)
print("Posterior Probability for Model (b):", post_b)
print("Posterior Probability for Model (c):", post_c)
```

Outputs given are :

Posterior Probability for Model (a): 9.142986210361563e-255
Posterior Probability for Model (b): 1.4339011785434019e-188
Posterior Probability for Model (c): 1.0

3 EM for Binary Data

3.1

$\mathcal{D} = \{x^{(1)}, \ldots, x^{(N)}\}$ are i.i.d sampled from $P(\mathbf{x} \mid \mathbf{p})$. Let $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_K)$ be the mixing proportions random vector (unit vector), and $\boldsymbol{P} = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_K)$ be a matrix with rows equal to K parameter vectors, each associated with a component k. Thus, the likelihood is given by:

$$P(\mathcal{D} \mid \boldsymbol{\pi}, \boldsymbol{P}) = \prod_{n=1}^{N} P(\boldsymbol{x}^{(n)} \mid \boldsymbol{\pi}, \boldsymbol{P}) = \prod_{n=1}^{N} \left[\sum_{k=1}^{K} (\pi_k) P_k(\boldsymbol{x}^{(n)} \mid \boldsymbol{p}_k) \right]$$

Now, as the pixels are independent of each other within each component distribution, then:

$$P_k(\boldsymbol{x}^{(n)} \mid \boldsymbol{p}_k) = \prod_{d=1}^{D} p_{kd}^{x_d^{(n)}} (1 - p_{kd})^{(1 - x_d^{(n)})}$$

Thus,

$$P(\mathcal{D} \mid \boldsymbol{\pi}, \boldsymbol{P}) = \prod_{n=1}^{N} \left[\sum_{k=1}^{K} (\pi_k) \left(\prod_{d=1}^{D} p_{kd}^{x_d^{(n)}} (1 - p_{kd})^{(1 - x_d^{(n)})} \right) \right]$$

3.2

Introduce a discrete latent variable $s^{(n)} \in \{1, \ldots, K\}$ where $P(s^{(n)} = k \mid \boldsymbol{\pi}) = \pi_k$, i.e., $s^{(n)} \sim \text{Discrete}[\boldsymbol{\pi}]$.

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

with $P(s^{(n)} = k, \boldsymbol{x}^{(n)} \mid \boldsymbol{\pi}, \boldsymbol{P}) = P(\boldsymbol{x}^{(n)} \mid s^{(n)} = k, \boldsymbol{\pi}, \boldsymbol{P}) P(s^{(n)} = k \mid \boldsymbol{\pi}, \boldsymbol{P})$.

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

As we saw previously,

$$P(\mathbf{x}^{(n)} \mid s^{(n)} = k, \boldsymbol{\pi}, \boldsymbol{P}) = P_k(\boldsymbol{x}^{(n)} \mid \boldsymbol{p}_k) = \prod_{d=1}^{D} p_{kd}^{x_d^{(n)}} (1 - p_{kd})^{(1-x_d^{(n)})}$$

and,

$$P(s^{(n)} \mid \boldsymbol{\pi}, \boldsymbol{P}) = P(s^{(n)} = k \mid \boldsymbol{\pi}) = \pi_k$$

Thus, we get:

$$r_{nk} = \frac{\pi_k \prod_{d=1}^{D} p_{kd}^{x_d^{(n)}} (1 - p_{kd})^{(1-x_d^{(n)})}}{\sum_{i=1}^{K} \pi_i \prod_{d=1}^{D} p_{id}^{x_d^{(n)}} (1 - p_{id})^{(1-x_d^{(n)})}}$$

This gives the E-step for an EM algorithm as r_{nk} is the posterior probability of component k given observation $\boldsymbol{x}^{(n)}$ and parameters $\boldsymbol{\pi}$, $\boldsymbol{P}$.

3.3

Let $\boldsymbol{\Theta} = \{\boldsymbol{\pi}, \boldsymbol{P}\}$, and $\boldsymbol{S} = (\boldsymbol{s}^{(1)}, \ldots, \boldsymbol{s}^{(N)})$ where each component is an indicator vector $\boldsymbol{s}^{(n)}$ where $s^{(n)} = k$ is equivalent to having $s_j^{(n)} = 0$ for all $j \neq k$, $j \in [K]$, and $s_k^{(n)} = 1$ (means that $\boldsymbol{x}^{(n)}$ was generated by the kth component). This will allow us to write $P(\boldsymbol{x}^{(n)} \mid s^{(n)} = k, \boldsymbol{\Theta})$ in the form of a product:

$$P(\boldsymbol{x}^{(n)} \mid s^{(n)} = k, \boldsymbol{\Theta}) = \prod_{k=1}^{K} P_k(\boldsymbol{x}^{(n)} \mid \boldsymbol{p}_k)^{s_k^{(n)}}.$$

So the joint distribution is $P(\mathcal{D}, \boldsymbol{S} \mid \boldsymbol{\Theta}) = P(\mathcal{D} \mid \boldsymbol{S}, \boldsymbol{\Theta}) P(\boldsymbol{S} \mid \boldsymbol{\Theta})$, where the prior distribution on the latent variables is:

$$P(\boldsymbol{S} \mid \boldsymbol{\Theta}) = \prod_{n=1}^{N} \prod_{k=1}^{K} (\pi_k)^{s_k^{(n)}},$$

which is going to give us the product of N mixing densities.

$$\Rightarrow \log P(\mathcal{D}, \boldsymbol{S} \mid \boldsymbol{\Theta}) = \log P(\mathcal{D} \mid \boldsymbol{S}, \boldsymbol{\Theta}) + \log P(\boldsymbol{S} \mid \boldsymbol{\Theta})$$

$$= \log \left[\prod_{n=1}^{N} \prod_{k=1}^{K} (P_k(\boldsymbol{x}^{(n)} \mid \boldsymbol{p}_k))^{s_k^{(n)}} \right] + \log \left[\prod_{n=1}^{N} \prod_{k=1}^{K} (\pi_k)^{s_k^{(n)}} \right]$$

$$= \sum_{n=1}^{N} \sum_{k=1}^{K} s_k^{(n)} \left[\log P_k(\boldsymbol{x}^{(n)} \mid \boldsymbol{p}_k) + \log \pi_k \right]$$

$$\Rightarrow \log P(\mathcal{D}, \boldsymbol{S} \mid \boldsymbol{\Theta}) = \sum_{n=1}^{N} \sum_{k=1}^{K} s_k^{(n)} \left[\left(\sum_{d=1}^{D} x_d^{(n)} \log p_{kd} + (1 - x_d^{(n)}) \log(1 - p_{kd}) \right) + \log \pi_k \right]$$

Now, $q(\{\boldsymbol{s}^{(n)}\}) = P(\boldsymbol{s}^{(n)} \mid \boldsymbol{x}^{(n)}, \boldsymbol{\Theta})$, and we know that the expectation of an indicator vector is equal to the vector of its probabilities.

$$\Rightarrow \langle \boldsymbol{s}^{(n)} \rangle_{P(\boldsymbol{s}^{(n)} \mid \boldsymbol{x}^{(n)}, \boldsymbol{\Theta})} = \left(P(s^{(n)} = 1 \mid \boldsymbol{x}^{(n)}, \boldsymbol{\Theta}), \ldots, P(s^{(n)} = K \mid \boldsymbol{x}^{(n)}, \boldsymbol{\Theta}) \right).$$

$$\Rightarrow \langle s_k^{(n)} \rangle_{P(\boldsymbol{s}^{(n)} \mid \boldsymbol{x}^{(n)}, \boldsymbol{\Theta})} = P(s^{(n)} = k \mid \boldsymbol{x}^{(n)}, \boldsymbol{\Theta}) = r_{nk}.$$

Observe that the rest of the terms in the sum are constant with respect to $P(\boldsymbol{s}^{(n)} \mid \boldsymbol{x}^{(n)}, \boldsymbol{\Theta})$, thus, by properties of expectation, the expected log-joint is equal to:

$$\sum_{n=1}^{N} \sum_{k=1}^{K} \langle s_k^{(n)} \rangle_{P(\boldsymbol{s}^{(n)} \mid \boldsymbol{x}^{(n)}, \boldsymbol{\Theta})} \left[\left(\sum_{d=1}^{D} x_d^{(n)} \log p_{kd} + (1 - x_d^{(n)}) \log(1 - p_{kd}) \right) + \log \pi_k \right]$$

$$\Rightarrow \langle \log P(\mathcal{D}, \boldsymbol{S} \mid \boldsymbol{\Theta}) \rangle_{P(\boldsymbol{S} \mid \mathcal{D}, \boldsymbol{\Theta})} = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \left[\left(\sum_{d=1}^{D} x_d^{(n)} \log p_{kd} + (1 - x_d^{(n)}) \log(1 - p_{kd}) \right) + \log \pi_k \right].$$

Now, to obtain an iterative update of parameters $\boldsymbol{\pi}, \boldsymbol{P}$ for the M-step, we need to find the arguments that maximize the expected log-joint.

Let's first start with $\boldsymbol{P}$. Thus, we differentiate the expected log-joint with respect to p_{mj} (so the $\log \pi_k$ vanishes), where $m \in [K]$ and $j \in [D]$, and set the derivative to zero to obtain:

$$\sum_{n=1}^{N} r_{nm} \frac{\partial}{\partial p_{mj}} \left(x_j^{(n)} \log(p_{mj}) + (1 - x_j^{(n)}) \log(1 - p_{mj}) \right) = 0, \quad \forall m, j \in [K], [D] \text{ respectively.}$$

$$\Leftrightarrow \sum_{n=1}^{N} r_{nm} \left(\frac{x_j^{(n)}}{p_{mj}} - \frac{(1 - x_j^{(n)})}{1 - p_{mj}} \right) = 0$$

$$\Leftrightarrow \sum_{n=1}^{N} r_{nm} x_j^{(n)} = \frac{p_{mj}}{1 - p_{mj}} \sum_{n=1}^{N} r_{nm} (1 - x_j^{(n)})$$

$$\Leftrightarrow \left(\sum_{n=1}^{N} r_{nm} x_j^{(n)} \right) \left(1 + \frac{p_{mj}}{1 - p_{mj}} \right) = \frac{p_{mj}}{1 - p_{mj}} \sum_{n=1}^{N} r_{nm}$$

$$\Rightarrow \hat{p}_{mj} = \frac{\sum_{n=1}^{N} r_{nm} x_j^{(n)}}{\sum_{n=1}^{N} r_{nm}}$$

$$\Rightarrow \hat{\boldsymbol{P}} = \left(\frac{\sum_{n=1}^{N} r_{ni} x_j^{(n)}}{\sum_{n=1}^{N} r_{ni}} \right)_{ij} \quad \text{for } i \in [K], j \in [D].$$

For $\boldsymbol{\pi}$, we have a constrained problem as $\sum_{k=1}^{K} \pi_k = 1$, so we will use a Lagrangian with a dual variable $\lambda \neq 0$. Now, at the maximum, the gradients of the expected log-joint with respect to $\boldsymbol{\pi}$ and the constraint are of opposite directions, so they have opposite signs. As we are not in vector form, solving this constrained problem is equivalent to solving the system of equations:

$$\frac{\partial \mathcal{L}(\boldsymbol{\pi}, \lambda)}{\partial \pi_j} = 0$$

$$\frac{\partial \mathcal{L}(\boldsymbol{\pi}, \lambda)}{\partial \lambda} = 0$$

Where,

$$\mathcal{L}(\boldsymbol{\pi}, \lambda) = \left(\sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \log(\pi_k) \right) - \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right)$$

We disregard terms involving probabilities p_{kd} as they are constant with respect to $\boldsymbol{\pi}$.

$$\frac{\partial \mathcal{L}(\boldsymbol{\pi}, \lambda)}{\partial \pi_j} = 0$$

$$\Leftrightarrow \sum_{n=1}^{N} \frac{r_{nj}}{\pi_j} - \lambda = 0$$

$$\Rightarrow \pi_j = \frac{\sum_{n=1}^{N} r_{nj}}{\lambda}$$

Now, plugging this result into $\frac{\partial \mathcal{L}(\boldsymbol{\pi}, \lambda)}{\partial \lambda} = 0$, we get

$$\frac{\partial}{\partial \lambda} \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \log \left(\frac{\sum_{n=1}^{N} r_{nk}}{\lambda} \right) - \frac{\partial}{\partial \lambda} \left(\sum_{k=1}^{K} \left(\sum_{n=1}^{N} r_{nk} \right) - \lambda \right) = 0$$

$$\Leftrightarrow \sum_{n=1}^{N} \sum_{k=1}^{K} -\frac{r_{nk}}{\lambda} + \lambda = 0$$

$$\Rightarrow \lambda = \sum_{n=1}^{N} \left(\sum_{k=1}^{K} r_{nk} \right)$$

$$\Rightarrow \pi_j = \frac{\sum_{n=1}^{N} r_{nj}}{\sum_{n=1}^{N} \left(\sum_{k=1}^{K} r_{nk} \right)} = \frac{\sum_{n=1}^{N} r_{nj}}{N}$$

as $\sum_{k=1}^{K} r_{nk} = 1$.

$$\Rightarrow \hat{\boldsymbol{\pi}} = (\frac{\sum_{n=1}^{N} r_{n1}}{N}, .., \frac{\sum_{n=1}^{N} r_{nK}}{N})$$

3.4

We will use small priors for our parameters and use EM to find the MAP estimates.

Posterior likelihood is $P(\boldsymbol{\Theta}|\mathcal{D}, \boldsymbol{S}) \propto P(\mathcal{D}, \boldsymbol{S}|\boldsymbol{\Theta})P(\boldsymbol{\Theta})$.

$$\begin{aligned} &= P(\mathcal{D}|\boldsymbol{S}, \boldsymbol{\Theta})P(\boldsymbol{S}|\boldsymbol{\Theta})P(\boldsymbol{\Theta}) \\ &= P(\mathcal{D}|\boldsymbol{S}, \boldsymbol{P})P(\boldsymbol{S}|\boldsymbol{\pi})P(\boldsymbol{\pi}, \boldsymbol{P}) \end{aligned}$$

Assuming priors on $\boldsymbol{\pi}$ and $\boldsymbol{P}$ are independent, then $P(\boldsymbol{\pi}, \boldsymbol{P}) = P(\boldsymbol{\pi})P(\boldsymbol{P})$.

$$\Rightarrow P(\boldsymbol{\Theta}|\mathcal{D}, \boldsymbol{S}) \propto P(\mathcal{D}|\boldsymbol{S}, \boldsymbol{P})P(\boldsymbol{S}|\boldsymbol{\pi})P(\boldsymbol{\pi})P(\boldsymbol{P})$$

$P(\boldsymbol{S}|\boldsymbol{\pi})$ follows a categorical distribution, so $P(\boldsymbol{\pi}) \sim \text{Dir}(\boldsymbol{\alpha})$ as the conjugate prior of the categorical distribution is the Dirichlet distribution. We assume no prior bias to the prior mixing weights (we only use priors as smoothing terms), so $\boldsymbol{\alpha} = (\alpha, \ldots, \alpha)$ where α is a small positive constant.

$$\Rightarrow P(\boldsymbol{\pi}) = \text{cst} \times \prod_{k=1}^{K} \pi_k^{\alpha-1}, \text{ where cst is the normalizing constant that will vanish when finding the arguments t}$$

Now, as $P(\mathcal{D}|\boldsymbol{S}, \boldsymbol{P}) = \prod_{n=1}^{N} \prod_{k=1}^{K} (P_k(\boldsymbol{x}^{(n)}|\boldsymbol{p}_k))^{s_k^{(n)}}$, and $P_k(\boldsymbol{x}^{(n)}|\boldsymbol{p}_k)$ is a product of Bernoulli distributions for each pixel of each component, the conjugate prior for each pixel probability of each component is a Beta distribution, say $B(\beta, \beta)$. Again, we assume no prior bias; this is just used for regularization. The beta prior will be, for all k, d:

$$P(p_{kd}) = \text{cst} \times p_{kd}^{\beta-1}(1 - p_{kd})^{\beta-1}$$

$$\Rightarrow \log \text{ posterior} \propto \log \left[\prod_{n=1}^{N} \prod_{k=1}^{K} \prod_{d=1}^{D} p_{kd}^{(s_k^{(n)} x_d^{(n)} + \beta - 1)} (1 - p_{kd})^{(s_k^{(n)}(1 - x_d^{(n)}) + \beta - 1)} \right] + \log \left[\prod_{n=1}^{N} \prod_{k=1}^{K} \pi_k^{(s_k^{(n)} + \alpha - 1)} \right]$$

(We discarded the normalizing constants of priors as well, as they vanish when differentiating the log posterior).

$$\Rightarrow \log \text{ posterior} \propto \log P(\mathcal{D}, \boldsymbol{S}|\boldsymbol{\Theta}) + \sum_{n=1}^{N} \sum_{k=1}^{K} (\alpha - 1) \log \pi_k + \sum_{n=1}^{N} \sum_{k=1}^{K} \sum_{d=1}^{D} (\beta - 1) \log p_{kd} + (\beta - 1) \log(1 - p_{kd})$$

Thus, by properties of expectation,

$$\langle \log \text{posterior} \rangle_{P(\boldsymbol{S}|\mathcal{D}, \boldsymbol{\Theta})} = \langle \log P(\mathcal{D}, \boldsymbol{S}|\boldsymbol{\Theta}) \rangle_{P(\boldsymbol{S}|\mathcal{D}, \boldsymbol{\Theta})} + \sum_{n=1}^{N} \sum_{k=1}^{K} (\alpha - 1) \log \pi_k$$

$$+\sum_{n=1}^{N}\sum_{k=1}^{K}\sum_{d=1}^{D}(\beta-1)\log p_{kd}+(\beta-1)\log(1-p_{kd})$$

Just like we saw in the slides, the expected sufficient statistic for MAP is the same as priors do not depend on latent variables.

Now let's compute MAP estimates for $\boldsymbol{\pi}$ and $\boldsymbol{P}$. For both parameters, the MAP estimates are derived analogously.

$$\frac{\partial \mathcal{L}_{MAP}(\lambda,\boldsymbol{\pi})}{\partial \pi_j}=0$$

$$\Leftrightarrow \sum_{n=1}^{N}\frac{r_{nj}}{\pi_j}+\frac{(\alpha-1)}{\pi_j}-\lambda=0$$

$$\Rightarrow \pi_j=\frac{\left(\sum_{n=1}^{N}r_{nj}\right)+(\alpha-1)}{\lambda}$$

Differentiating with respect to λ with $\pi_j=\frac{\left(\sum_{n=1}^{N}r_{nj}\right)+(\alpha-1)}{\lambda}$, we get

$$\frac{\partial}{\partial\lambda}\left[\sum_{n=1}^{N}\sum_{k=1}^{K}r_{nk}\log\frac{\left(\sum_{n=1}^{N}r_{nj}\right)+(\alpha-1)}{\lambda}-\sum_{k=1}^{K}(\alpha-1)\log\frac{\left(\sum_{n=1}^{N}r_{nj}\right)+(\alpha-1)}{\lambda}\right]$$

$$+\frac{\partial}{\partial\lambda}\left[\left(\sum_{k=1}^{K}\left(\sum_{n=1}^{N}r_{nj}\right)+(\alpha-1)\right)-\lambda\right]=0$$

$$\Rightarrow \lambda=N+K(\alpha-1)$$

$$\Rightarrow \hat{\pi}_j=\frac{\left(\sum_{n=1}^{N}r_{nj}\right)+\alpha-1}{N+K(\alpha-1)},\quad \text{for } j\in[K]$$

Now, for p_{mj} we will replace $\beta-1$ with β for simplicity in notation.

$$\frac{\partial}{\partial p_{mj}}\langle\log\text{posterior}\rangle_{P(\boldsymbol{S}|\mathcal{D},\boldsymbol{\Theta})}=0$$

$$\Leftrightarrow \sum_{n=1}^{N} r_{nm}\left(\frac{x_j^{(n)}}{p_{mj}} - \frac{(1-x_j^{(n)})}{1-p_{mj}}\right) + \left(\frac{\beta}{p_{mj}} - \frac{\beta}{1-p_{mj}}\right) = 0$$

$$\Leftrightarrow \sum_{n=1}^{N} r_{nm}x_j^{(n)} - \frac{p_{mj}}{1-p_{mj}}\sum_{n=1}^{N} r_{nm} + \frac{p_{mj}}{1-p_{mj}}\sum_{n=1}^{N} r_{nm}x_j^{(n)} + \beta - \frac{\beta p_{mj}}{1-p_{mj}} = 0$$

$$\Leftrightarrow \frac{1}{1-p_{mj}}\sum_{n=1}^{N} r_{nm}x_j^{(n)} - \frac{p_{mj}}{1-p_{mj}}\sum_{n=1}^{N} r_{nm} + \beta - \frac{\beta p_{mj}}{1-p_{mj}} = 0$$

$$\Leftrightarrow \sum_{n=1}^{N} r_{nm}x_j^{(n)} - p_{mj}\sum_{n=1}^{N} r_{nm} + \beta - 2p_{mj}\beta = 0$$

$$\Leftrightarrow p_{mj} = \frac{\sum_{n=1}^{N} r_{nm}x_j^{(n)} + \beta}{\sum_{n=1}^{N} r_{nm} + 2\beta}$$

Now, coming back to hyperparameter $\beta - 1$, we get

$$\hat{p}_{mj} = \frac{(\sum_{n=1}^{N} r_{nm}x_j^{(n)}) + \beta - 1}{(\sum_{n=1}^{N} r_{nm}) + 2(\beta - 1)}$$

for all $m \in [K]$, $j \in [D]$.

As you can see the MAP estimate for mixing weights does sum up to 1, but the MAP estimate for probability vector of each component doesn't This is because we aren't taking into account the normalizing constants when calculating the log posterior, so the MAP estimate to probability distributions of components is just the unnormalized posterior probability matrix.

Code for EM algorithm :

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import beta, dirichlet
from scipy.special import logsumexp

def initialize_parameters(K, D):
    # Initialize mixing coefficients pi, with uniform probability.
    pi = (1 / K) * np.ones(K)

    # Initialize parameters P for each component
    # P is initialized to be in the range [0.1, 0.9]
```

```
    # This avoids values too close to 0 or 1 which would lead to numerical instabilities in the log terms
    P = 0.1 + 0.8 * np.random.rand(K, D)
    return pi, P

def calculate_log_prior(P, pi, alpha_P= 1 + 1e-10 , beta_P= 1 + 1e-10):
    # Compute log priors for P and pi
    log_prior_P = 0
    for k in range(P.shape[0]):
        for d in range(P.shape[1]):
            log_prior_P += beta.logpdf(P[k, d], alpha_P, beta_P)
    log_prior_pi = dirichlet.logpdf(pi, (1 + 1e-10) * np.ones(len(pi)))
    return log_prior_P, log_prior_pi

def calculate_responsibilities(X, P, pi, epsilon=1e-10):
    N, D = X.shape
    K = pi.shape[0]

    #add a samll epsilon to avoid very small pi a p values in log to avoid numerical instabilities when calculating
    #responsibilities in future iterations E step. We already dealt with that at initialization, but we also need to deal with
    #it at each iteration as obviously pi and p values change through multiple iterarions.

    # Broadcasting operations to calculate log probabilities
    log_P = np.log(P + epsilon)
    log_P_complement = np.log(1 - P + epsilon)

    # Calculate the log responsibilities for all data points and components at once using matrix operations
    log_R = X @ log_P.T + (1 - X) @ log_P_complement.T + np.log(pi + epsilon)

    # Normalize using log-sum-exp to prevent underflow/overflow
    log_R -= logsumexp(log_R, axis=1)[:, np.newaxis]

    # Convert back to actual probabilities
    R = np.exp(log_R)

    return R

def log_likelihood(X, pi, P):
    # Compute the log probability of each pixel being 1 for all images and components
    #Again add samll epsilon to avoid numerical issues
    log_prob_on = X @ np.log(P.T + 1e-10)
    log_prob_off = (1 - X) @ np.log(1 - P.T + 1e-10)
    log_prob = log_prob_on + log_prob_off

    # Add the log probability of each component
    log_prob += np.log(pi + 1e-10)

    # Compute the log likelihood using log-sum-exp
    log_likelihood = logsumexp(log_prob, axis=1).sum()
```

```
    return log_likelihood


def EM_algorithm_with_priors(K, X, max_iterations=40, tolerance=1e-6, alpha_P=1 + 1e-10, beta_P=1 + 1e-10, alpha_pi=1 + 1e-10):
    N, D = X.shape
    pi, P = initialize_parameters(K, D)
    prev_log_posterior = float('-inf')
    responsibilities_history = []
    P_history = []
    log_posteriors = []
    pi_history = []

    for iteration in range(max_iterations):
        # E-step: Calculate responsibilities using the previous iteration M step
        responsibilities = calculate_responsibilities(X, P, pi)
        responsibilities_history.append(responsibilities)

        # M-step
        # Introduce priors in the updates
        # Update for P
        numerator = responsibilities.T @ X + (alpha_P - 1)
        denominator = np.sum(responsibilities, axis=0)[:, np.newaxis] + (2 * alpha_P - 2)
        P = numerator / denominator

        # Update for pi
        pi = (responsibilities.sum(axis=0) + alpha_pi -1) / np.sum((responsibilities.sum(axis=0) + alpha_pi -1), axis=0)
        pi_history.append(pi)
        P_history.append(P)

        # Calculate log likelihood
        log_likelihood_value = log_likelihood(X, pi, P)

        # Calculate log prior
        log_prior_P, log_prior_pi = calculate_log_prior(P, pi, alpha_P, beta_P)

        # Calculate log posterior
        log_posterior = log_likelihood_value + log_prior_P + log_prior_pi
        log_posteriors.append(log_posterior)

        # Check for convergence based on log posterior
        if log_posterior - prev_log_posterior < tolerance:
            break
        prev_log_posterior = log_posterior

    return pi, P, responsibilities, P_history, pi_history, log_posteriors

K_values = [2, 3, 4, 7, 10]

pi_values_for_K_values = []
P_values_for_K_values = []
```

```
log_posteriors_for_K_values = []
final_responsibilities_for_K_values = []

for K in K_values:
    print(f"Running EM algorithm for K = {K}")
    pi, P, responsibilities, P_history, pi_history,
    log_posterior_history = EM_algorithm_with_priors(K, X)
    pi_values_for_K_values.append(pi)
    P_values_for_K_values.append(P)
    log_posteriors_for_K_values.append(log_posterior_history)
    final_responsibilities_for_K_values.append(responsibilities)

# Create a single figure with subplots
plt.figure(figsize=(12, 8))

for i in range(len(K_values)):
    plt.subplot(2, 3, i+1)  # Create a subplot for each K value
    plt.plot(log_posteriors_for_K_values[i])
    plt.xlabel('Iteration')
    plt.ylabel('Log Posterior')
    plt.title(f'Log Posterior for K={K_values[i]}')

plt.tight_layout()
plt.show()
```

log-posterior as a function of the iteration number:

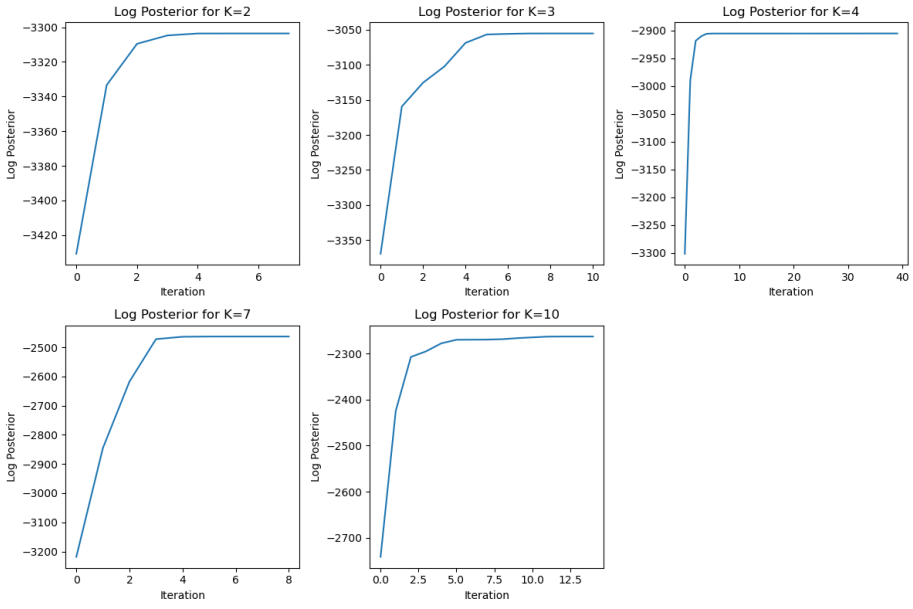

Figure 3: Log posteriors as a function of iteration number

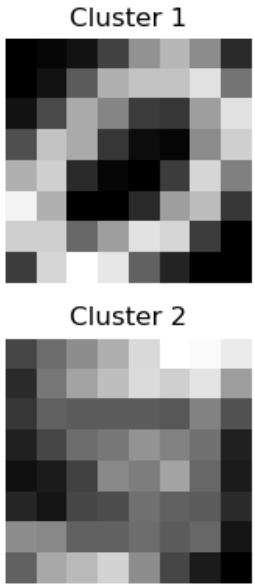

Figure 4: K=2

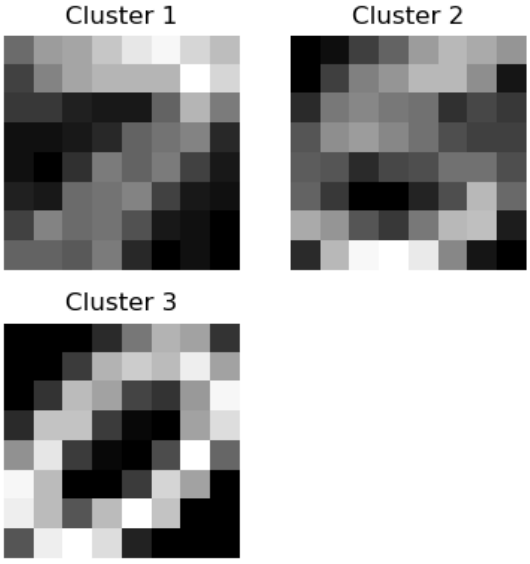

Figure 5: K=3

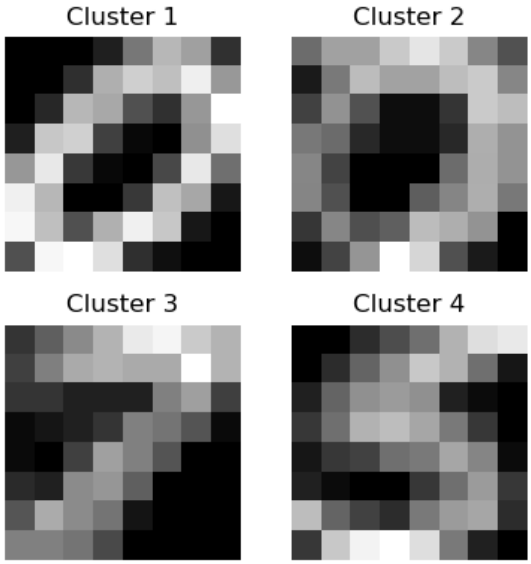

Figure 6: K=4

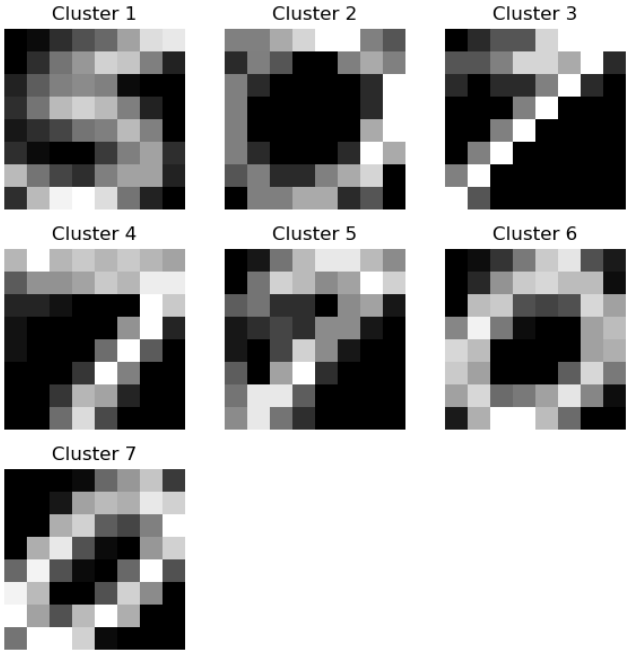

Figure 7: K=7

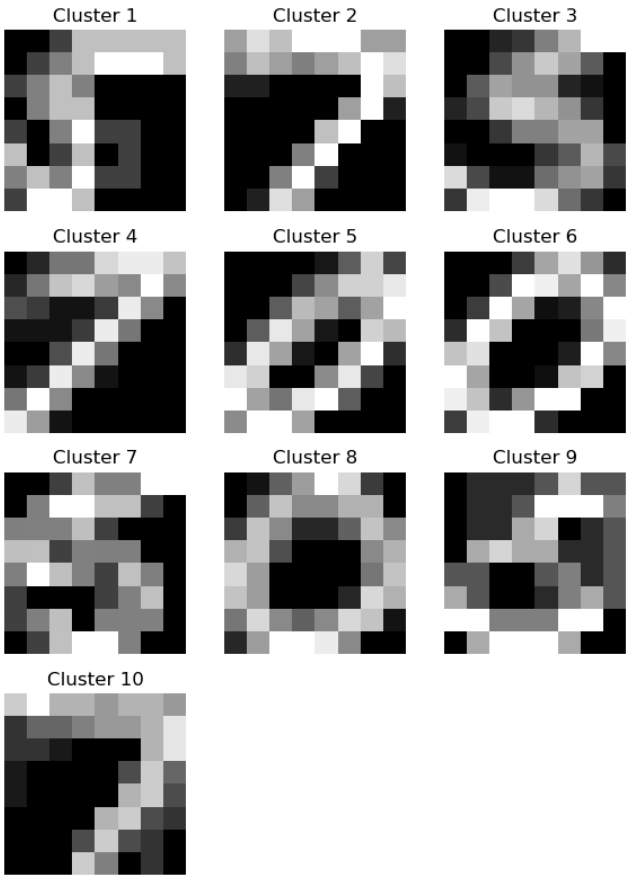

Figure 8: K=10

Corresponding cluster weights (kth term in list corresponds to cluster k):

K=2 : [0.41963453, 0.58036547]

K=3 : [0.34004724, 0.36001849, 0.29993427]

K=4 : [0.31906035, 0.21217701, 0.23876309, 0.22999955]

K=7 : [0.22 , 0.06 , 0.06 , 0.14 , 0.11 , 0.1898866, 0.2201134]

K=10 : [0.04 , 0.08 , 0.14 , 0.13 , 0.11000037, 0.16997304, 0.04 , 0.1300165 , 0.0600101 , 0.1]

3.5

By running this code:

```
num_runs = 14  # Number of times you want to run the EM algorithm
K_values = [2, 3, 4, 7, 10]  # Different values of K to vary

results = []
for _ in range(num_runs):
    for K in K_values:
        pi, P, _, _, _, log_likelihood_history =
    EM_algorithm_with_priors(K, X)
        results.append((K, pi, P, log_likelihood_history))
```

Here are the learned probability vectors as images for each K from two different random initializations taken from the code above:

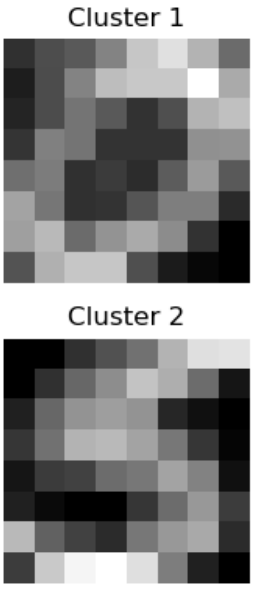

Figure 9: K=2

Figure 10: K=2 different initialization

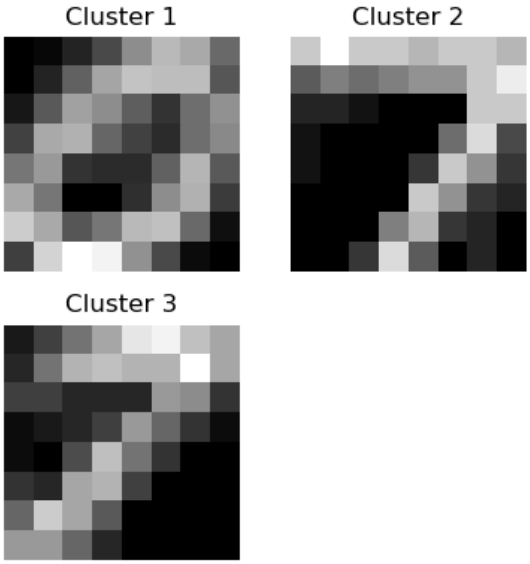

Figure 11: K=3

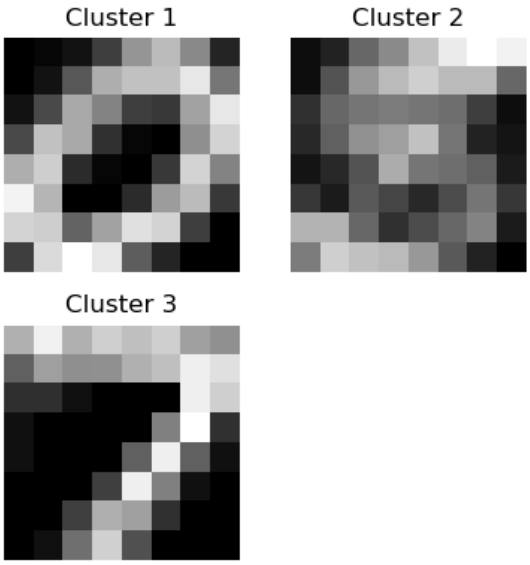

Figure 12: K=3 different initialization

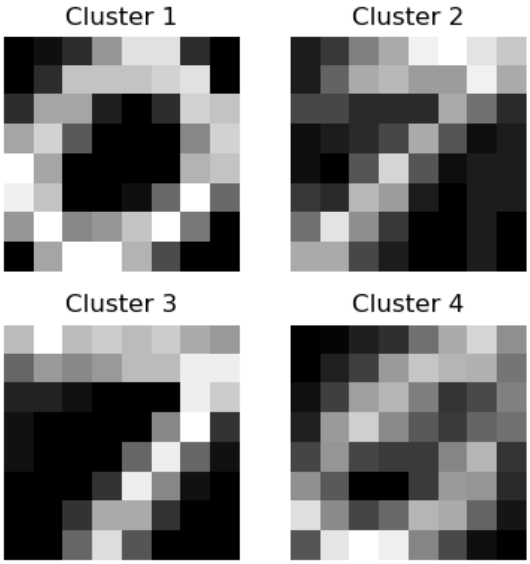

Figure 13: K=4

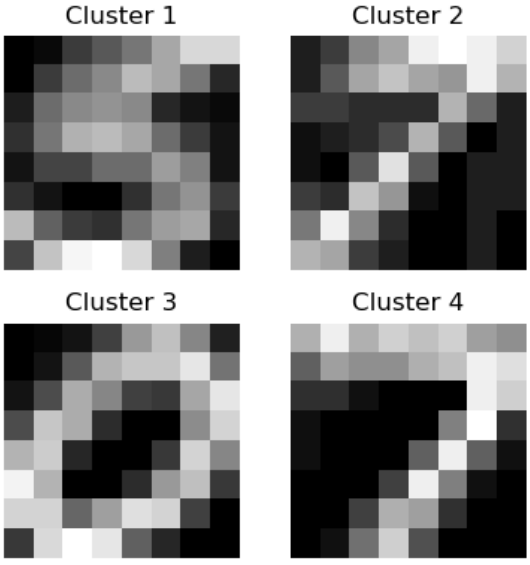

Figure 14: K=4 different initialization

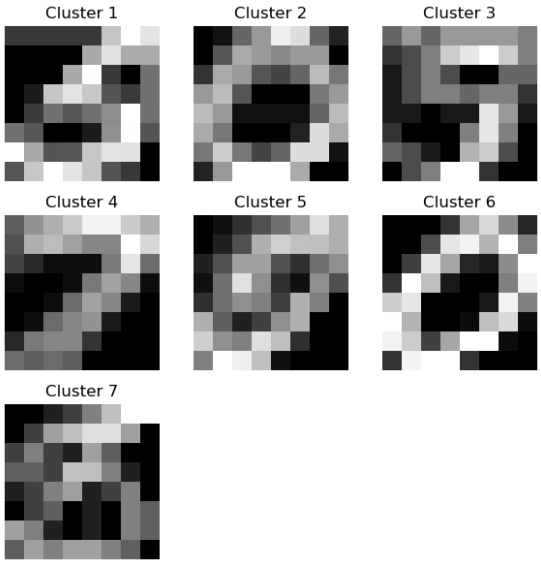

Figure 15: K=7

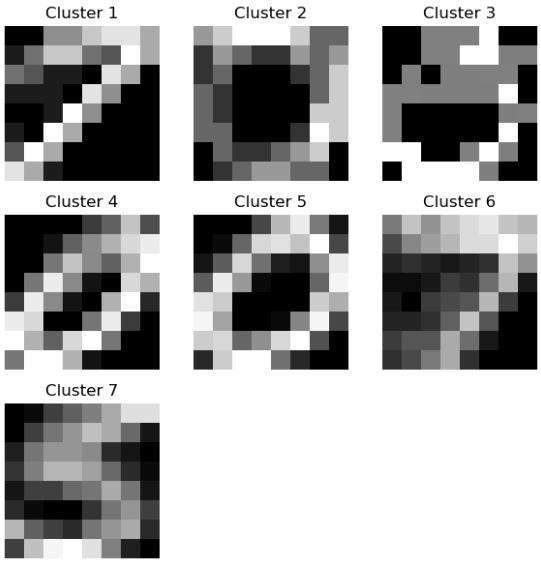

Figure 16: K=7 different initialization

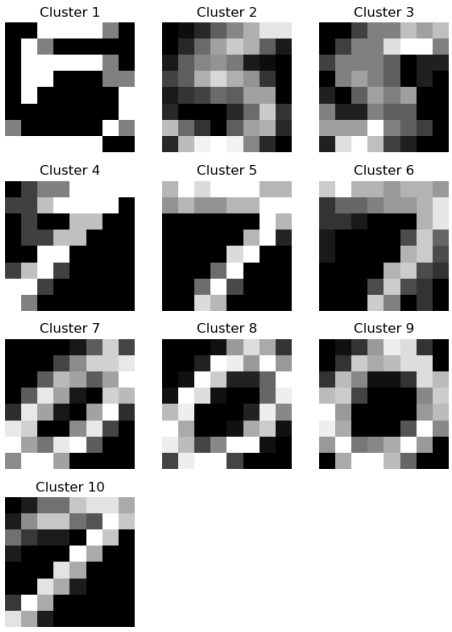

Figure 17: K=10

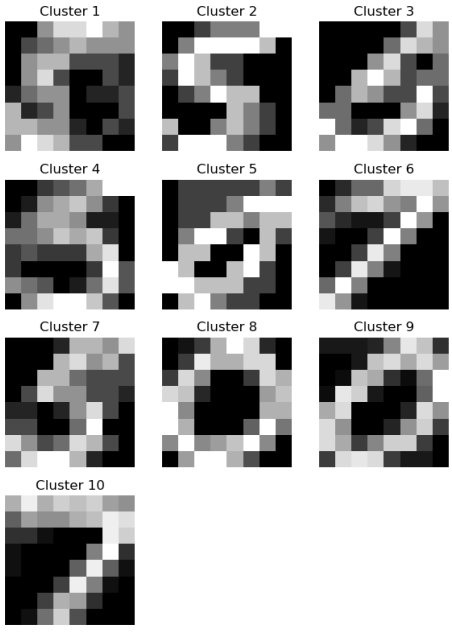

Figure 18: K=10 different initialization

Consistency of Solutions up to permutation:

In examining the outcomes of the EM algorithm for different values of K, a key observation is that the solutions are not the same up to permutation, and this discrepancy is notably influenced by the choice of K. When running the EM algorithm multiple times with different initializations, especially for lower K values like 2, 3, and 4, the solutions or clusters tend to oscillate between a few main shapes, including diagonal staircases, ovals, circles and S-like formations. This pattern suggests that while the overall nature of the solutions is somewhat consistent, the specific shapes present in the clusters can vary depending on the initialization. Essentially, the form of the solution shows minor variations, with the same set of 4-5 main shapes emerging in different permutations across various initializations.

As K increases, the observed behavior changes. At these higher values, the primary shapes are consistently represented in the clusters, albeit with slight variations. However, more uncommon shapes, which are often extreme variations of the main shapes (diagonal staircases, ovals, circles, S) begin to appear. These rarer shapes contribute to the solutions for higher K's not being equivalent up to permutation, as their presence in the clusters can change depending on the initialization. This observation highlights an interesting aspect of the algorithm's behavior: with an increase in K, the solutions exhibit a broader range of patterns, reflecting more diversity and complexity in the data set.

Cluster quality and interpretability:

For K=2 and K=3, the algorithm shows relative stability, consistently identifying primary shapes in the dataset across different runs. This stability, however, starts to diminish as K increases beyond 3. Higher values of K lead to more variability in solutions across different initializations, indicating an increase in model variance and sensitivity to the initial conditions. The choice of K significantly influences the nature of the clusters. Lower values of K tend to capture the most prominent and generalized features in the dataset but may overlook finer details, leading to bias. Conversely, higher values of K result in more specialized and distinct clusters, capturing subtler patterns and variations. However, this comes with the risk of overfitting to noise and outlier patterns. This can be seen in the corresponding mixing weights where these clusters usually have much lower assigned weight.

The quality and interpretability of the clusters reveal that prominent shapes in the dataset are generally well captured, particularly when K is around 3 or 4. As K increases, clusters become more specific and detailed. For instance, K values of 7 and 10 lead to clusters that not only capture the primary shapes clearly but also show minor variations and outliers, hinting at a tendency toward overfitting.

With $K \geq 4$, certain shapes appear to be represented multiple times with subtle

differences, indicating possible redundancy. This aspect underscores the need for balancing the choice of K to prevent excessive granularity that might not add significant value to the interpretation of the data.

Possible model improvements:

In terms of improvements, considering the model's sensitivity to initial conditions, particularly at higher K values, exploring different initialization strategies could be beneficial. Introducing regularization might help mitigate overfitting, a risk that becomes more pronounced with higher K values.

Additionally, addressing noise in the data is crucial. In the current EM setup, each pixel of each cluster is weighted equally in the beta prior, leading to noisy clusters (blurry outlines of shapes in clusters), especially for lower K's. One potential improvement could be the use of a Hidden Markov Random Field prior to the pixel probabilities of each cluster, multiplied with the joint beta prior used for the pixel probabilities. This approach would encourage neighboring pixels to have similar values (Image segmentation), thereby smoothing the cluster images, and making the model more robust. This could solve the problem for K=4, K=5 where the main shapes are captured as clusters, but are still slightly blurry (contours are not well defined). This could allow us to use mid-range K's (K=4, K=5) where model variance is low, and thus get a more robust model.

3.6

Last iteration Log-likelihoods obtained expressed in bits (absolute values):

K=2 : 4766.07379178

K=3 : 4357.37951432

K=4 : 4179.27535218

K=7 : 3639.46188353

K=10 : 3415.64382131

The length of the naive encoding of these binary data is $64 * 100 = 6400$ and the gzip compression size is 5418.

The EM algorithm's log-likelihood suggests that the data can be described using a total of $\sim$ 4800 to 3400 bits, while naive encoding would require 6400 bits. The magnitude log-likelihood in bits represents the number of bits that the model needs to encode the dataset. Thus, if this magnitude is lower than the length of the naive encoding then the model describes the data set using less information. This means that the model captures patterns/shapes present in the dataset and compresses them into clusters which effectively reduces the amount of information needed to explain the dataset. In summary, this implies

that the model effectively captured the underlying structure (i.e. probability distribution) of the data. Now, the output of gzip is a compressed file that includes all the necessary information to fully reconstruct the original data. It finds byte-level redundancies and patterns to compress the data using a combination of Huffman coding and LZ77. Here, gzip compression size results in 5418 bits. All of our log-likelihoods are lower in magnitude. This implies that in theory, our model, for each K, uses less information than gzip compression to describe the data. This difference is due to the fact that our model (Mixture of Multivariate Bernoulli) is tailored for this specific dataset. gzip is a compression algorithm that has a general use for all kinds of files and might not be the most suitable option for compression of this dataset.

While the log-likelihood in bits quantifies the efficiency of our model in describing the dataset within a theoretical framework, gzip compression yields an actual compressed file ready for practical use. The former offers insight into the statistical properties of the data as captured by the model, presenting a measure of potential compression. In contrast, gzip's output is a tangible object, i.e a compressed file that embodies the actual application of a compression algorithm. Therefore, when comparing the two, it's crucial to recognize that one reflects a model's theoretical compression capability, whereas the other provides a concrete, utilizable compression result. This fundamental difference implies that the comparison, while informative about the relative efficiency of data representation, does not involve directly equivalent entities: one is an idealized estimate, the other a practical implementation.

3.7

Total cost in bits for encoding both the model and the data for K's:

K=2 : 5026.07379178

K=3 : 4747.37951432

K=4 : 4699.27535218

K=7 : 4549.46188353

K=10 : 4715.64382131

The total bit cost for encoding both the model and the data across all considered K values remains below that of gzip compression. This indicates that our model is achieving a more effective compression for K=2,3,4,7, and 10, which is reasonable given that it is specifically designed for this data type, so even if we get poor results for K=2 for example, we still manage to identify clear patterns in the data (i.e 2 main shapes as clusters).

On the other hand, observe that the relationship between total cost and K is not strictly linear. Interestingly, we find that the total cost dips and then rises, reaching its most efficient point at K=7. This suggests that a K=7 model may

be optimally capturing the dataset structure without unnecessary complexity. This observation underscores the delicate balance between a model's complexity and its performance. A linear decrease in log-likelihood with increasing K does not always equate to improved model performance, as it may also signal overfitting. A metric that assesses how well a model explains its training data is not the sole indicator of its overall performance. Our visual analysis of cluster means indicates that an optimal K lies between 4 and 7, corroborated by the minimal total encoding cost at K=7. It appears that the cost of encoding the model parameters, combined with the log-likelihood, could serve as an effective metric for determining the optimal K. The encoding cost effectively introduces a penalty for model complexity, favoring models that strike a balance between fitting the data and maintaining simplicity. Therefore, we could employ an encoding-based evaluation metric to identify the most suitable K for our model by encoding the final iteration log-likelihoods and the model parameters into bits and then comparing the sums across different K values and choosing a K with minimum total cost in bits for encoding both parameters and log-likelihood. This approach offers a promising solution to the challenge of assessing overall model performance in unsupervised settings, where determining the optimal hyperparameter K can't be reliably achieved through visual inspection of cluster means alone, as is the case in this context.

4 Decrypting Messages with MCMC

4.1

To model English text as a Markov chain and estimate the transition probabilities $\psi(\alpha, \beta)$ and the stationary distribution $\phi(\gamma)$, we follow these steps:

Transition Probabilities $\psi(\alpha, \beta)$: The transition probability $\psi(\alpha, \beta)$ is the probability of transitioning from symbol β to symbol α. It can be estimated by counting the occurrences of each symbol pair and normalizing by the total occurrences of the preceding symbol. The formula for the maximum likelihood estimate is:

$$\psi(\alpha, \beta) = \frac{\text{Count}(\beta \to \alpha)}{\sum_{\gamma \in S} \text{Count}(\beta \to \gamma)}$$

where $\text{Count}(\beta \to \alpha)$ is the number of times symbol α follows symbol β in the text, and S is the set of all symbols.

Stationary Distribution $\phi(\gamma)$: The stationary distribution $\phi(\gamma)$ represents the long-term probability of observing each symbol γ. It can be estimated by counting the frequency of each symbol and normalizing by the total number of symbols. The formula is:

$$\phi(\gamma) = \frac{\text{Count}(\gamma)}{\sum_{\delta \in S} \text{Count}(\delta)}$$

where $\text{Count}(\gamma)$ is the number of occurrences of symbol γ, and S again represents the set of all symbols.
For both estimates, we can use a large corpus of English text, such as the English translation of 'War and Peace', to obtain reliable counts.

We will show the initial probabilities as a table and transition matrix as a coloured squared image. As most of the probabilities in the transition matrix are either 0 or very small it is better to represent it in logarithm scale. (We will add a small constant of $\exp(-18)$ to any probability lower than that values to avoid $\log(0)$).

Empirical transition probabilities:

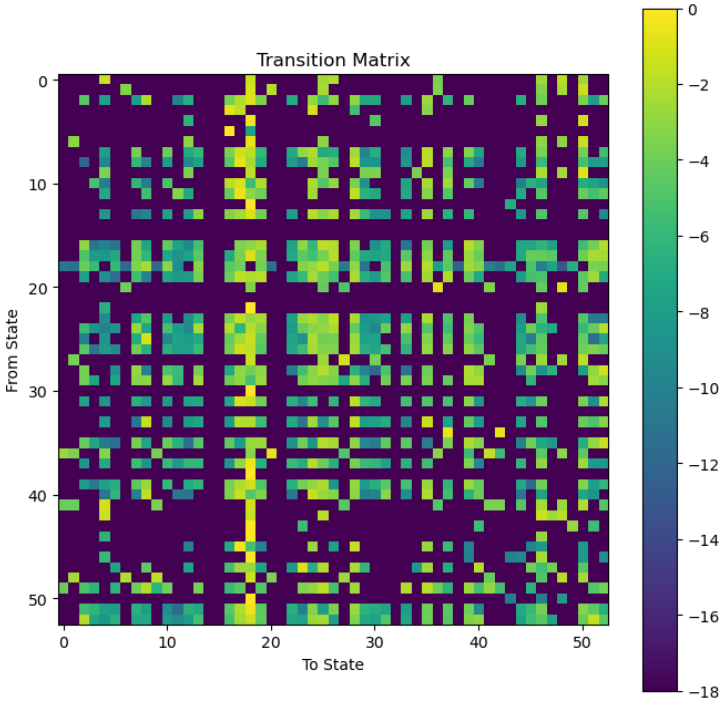

Figure 19: Empirical transition probabilities

Empirical stationary distribution:

Symbol	Probability
3	1.374334779018993e-05
7	1.06510945373971 96e-05
g	0.0165301551383456 94
j	0.0007943655022729 78
)	0.0001944683712311875
q	0.0007620686349660315
9	7.902424979359209e-06
m	0.01940491991235867
h	0.0534990735265671
5	1.2369013011170937e-05
b	0.010765851491445282
z	0.0004439101336231 3474
:	0.0003284660121855 3933
f	0.018055323159362 02
]	0.0
[	0.0
u	0.020353898077271285
e	0.10105792855451934
	0.18127029076457335
l	0.030060824621482 43
8	5.291188899223123e-05
'	0.0
!	0.0005232779671114816
p	0.014217149705256728
o	0.059508008764132 88
t	0.072005865660836 86
r	0.046570708321837594
6	1.4086931484944677e-05
i	0.054992631847665986
-	0.000518467795384915
;	0.0003721011414193923 7
k	0.005557122678963298
"	0.0
w	0.019069238642583 28
/	6.871673895094965e-07
a	0.063866024348402 1
1	9.551626714182002e-05
y	0.013981451290654 97
=	6.871673895094965e-07
s	0.049864988787146125
c	0.019445462788339 73
2	3.882495750728655e-05
4	4.8101717265664756e-06
*	9.139326280476303e-05
?	0.0004329154553909 8276
v	0.007062019261989 0955
.	0.008670334537136072
x	0.001302525786815 2507
0	4.4322296623362523e-05
(	0.0002054630494633 3945
,	0.011462639224407 91
d	0.038413687824665116
n	0.058048465228814 72

Figure 20: Empirical stationary distribution

We will explain in question d) (i.e, section 4.4 here) why some of the initial probabilities are 0 here.

4.2

The latent variables $\sigma(s)$ for different symbols s are not independent as the latent variables form a permutation which is a one-to-one mapping. Thus the mapping of $\sigma(s)$ affects the mapping of other symbols. Specifically, if a symbol s_1 is mapped to an encrypted symbol e_1 (i.e., $\sigma(s_1) = e_1$), it directly influences the mapping possibilities of other symbols, as no other symbol s_2 can be mapped to e_1 due to the one-to-one nature of the mapping. This introduces a dependency among the latent variables: knowing the mapping of one symbol informs us about the mappings of others. For instance, if we know that $\sigma(a) = s$, we immediately know that no other symbol can be mapped to 's', thereby reducing the possible mappings for all other symbols.

Consider an encrypted English text represented as a sequence of symbols $e_1 e_2 \dots e_n$. Given a permutation σ, the joint probability of observing the encrypted sequence given the permutation σ can be written as:

$$P(e_1 e_2 \dots e_n | \sigma) = P(\sigma^{(-1)}(e_1)) \prod_{i=2}^{n} P(\sigma^{(-1)}(e_i) | \sigma^{(-1)}(e_{i-1}))$$

Where $\sigma^{(-1)}$ is the inverse mapping, which exists as σ is bijective and $\sigma^{(-1)}(e_i)$ is the decrypted symbol which mapped to e_i under σ. Thus, given the permutation σ we can calculate the probability of occurrence of an encrypted text by inverse mapping it and then calculating the likelihood of the decrypted text under $\sigma^{(-1)}$, which is known, using the Markov property given by model assumption.

$$\Rightarrow P(e_1 e_2 \dots e_n | \sigma) = \phi(\sigma^{(-1)}(e_1)) \prod_{i=2}^{n} \psi(\sigma^{(-1)}(e_i) | \sigma^{(-1)}(e_{i-1}))$$

4.3

In a list of n symbols there are $\binom{n}{2}$ possible distinct pairs of symbols thus $\frac{2}{n(n-1)}$ probability of choosing one of these pairs at random (assuming all symbols in symbol list are distinct).

$$\Rightarrow S(\sigma \to \sigma') = S(\sigma' \to \sigma) = \frac{2}{n(n-1)} \text{ for all pairs } (\sigma, \sigma') \text{ possible}$$

Thus the proposal does not depend on σ, σ', it is the same for all swapping possible. It is symmetric with uniform probability $\frac{2}{n(n-1)}$. This simplifies the rejection kernel for MH algorithm to :

$$A(\sigma \to \sigma') = \min\{1, \frac{\pi(\sigma')}{\pi(\sigma)}\}$$

Where π is the target distribution, i.e the posterior distribution of σ given encrypted text. i.e :

$$\frac{\pi(\sigma')}{\pi(\sigma)} = \frac{P(\sigma' | e_1 e_2 \dots e_n)}{P(\sigma | e_1 e_2 \dots e_n)} = \frac{\frac{P(\sigma', e_1 e_2 \dots e_n}{P(e_1 e_2 \dots e_n)}}{\frac{P(\sigma', e_1 e_2 \dots e_n)}{P(e_1 e_2 \dots e_n)}} = \frac{P(\sigma', e_1 e_2 \dots e_n)}{P(\sigma, e_1 e_2 \dots e_n)} = \frac{P(e_1 e_2 \dots e_n | \sigma') P(\sigma')}{P(e_1 e_2 \dots e_n | \sigma) P(\sigma)}$$

We assume uniform prior over permutations so $P(\sigma') = P(\sigma)$ for all (σ', σ) possible. Thus

$$A(\sigma \to \sigma') = \min\{1, \frac{P(e_1 e_2 \dots e_n|\sigma')}{P(e_1 e_2 \dots e_n|\sigma)}\}$$

Where,

$$\frac{P(e_1 e_2 \dots e_n|\sigma')}{P(e_1 e_2 \dots e_n|\sigma)} = \frac{\phi(\sigma'^{(-1)}(e_1))}{\phi(\sigma^{(-1)}(e_1))} \prod_{i=2}^{n} \frac{\psi(\sigma'^{(-1)}(e_i)|\sigma'^{(-1)}(e_{i-1}))}{\psi(\sigma^{(-1)}(e_i)|\sigma^{(-1)}(e_{i-1}))}$$

Now remark that by choosing two symbols s and s' at random and swapping the corresponding encrypted symbols $\sigma(s)$ and $\sigma'(s)$, then all of the terms that aren't affected by this swap will vanish in the expression $\frac{P(e_1 e_2 \dots e_n|\sigma')}{P(e_1 e_2 \dots e_n|\sigma)}$ as $\sigma(\alpha) = \sigma'(\alpha)$ for all α's in the Symbol list, with $\alpha \neq s, s'$.This provides a more efficient and stable way to compute the likelihood ratio.

4.4

In this section, we will work with log to provide a more computationally stable way of calculating the likelihood ratio as likelihoods tend to be very small leading to numerical instabilities, especially if we compute their ratio.

$$\Rightarrow \log A(\sigma \to \sigma') = \min\{0, \log \pi(\sigma') - \log \pi(\sigma)\}$$

Note that I will answer to question e) in this section as well. The complete code is given in the appendix. This code is split into two parts : **the code before applying the MCMC sampling, i.e how we treat the problem** and **The code for the MH algorithm**.

Pre-treatment of data in War and Peace:

In the original text, the symbol " appears in a different font style and is consequently not recognized as a valid symbol. This should not significantly impact our analysis, as the frequency of " in 'War and Peace' does not accurately reflect its stationary probability. This discrepancy arises because 'War and Peace' is a novel abundant in dialogue, leading to an unusually high occurrence of " compared to a typical English text. Furthermore, we have removed the header and additional comments at the end of the text to preserve only the novel's content. These sections often contain non-English words, links, copyright statements, and numerous numerals that do not represent standard English text frequency, essentially acting as noise. Also due to the symbol list given we do not distinguish between upper and lowercase letters so we transform every upper-case letter to lowercase in War and Peace.

Adding a small constant to restore ergodicity and deal with numerical instabilities:

Some typographical and punctuation symbols are absent in the text, resulting in a zero empirical frequency, which is not a true reflection of their presence in English. The same issue extends to transitions, where a significant portion (more than half of the 2809 possible transitions) also exhibit zero empirical frequency. While it might be reasonable for certain transitions like *) to have a negligible probability in English, this assumption compromises ergodicity. For instance, any permutation decrypting a text with a * to) transition would have zero likelihood, rendering the chain non-irreducible and non-aperiodic (e.g., the kk transition has zero probability). To address this, we add a small constant of $\exp(-18)$ (a rounded value of log(1e-8)) to all entries with zero probabilities in both the empirical stationary and transition distributions. This approach is viable as the Metropolis-Hastings acceptance criteria do not necessitate normalizing these probabilities, and the constant is sufficiently small to simulate a near-zero probability. By implementing this method we hit two birds with one stone : first, we restore erdigocity by making the transition matrix aperiodic and irreducible (making it probabilistically possible, although extremely unlikely) to transition between any two permutations possible, and secondly, we deal with the occurrence of log(0) and log of extremely small probabilities when computing the log ratio which would normally lead to nan values and numerical instabilities.

Intelligent initialization of chain:

Our initialization function for the decryption chain begins by aligning the symbol frequencies in the encrypted text with those observed in 'War and Peace', thereby approximating the typical frequencies of English symbols. This method involves creating an initial permutation that maps the most frequent symbols in the encrypted text to the most frequent symbols found in 'War and Peace'. Using a frequency-based permutation acknowledges the varying symbol frequencies inherent in English, making it a more informed and effective starting point than using an identity permutation. However, it's important to recognize that this method is not without its limitations. The encrypted text's smaller size compared to 'War and Peace' means that an exact match in frequency distribution is improbable. Additionally, the encrypted text may not include all symbols present in the symbol list, and due to its context-dependent nature, the frequencies of decrypted letters might not align perfectly with standard English. This variation is a consequence of the smaller dataset size, leading to higher variability and a reduced Effective Sample Size (ESS) when compared to more extensive text samples. Nonetheless, it provides a stronger, more data-driven foundation for the Metropolis-Hastings algorithm to begin its decryption process.

Output of MH algorithm after 7500 iterations:

Iter 0: on lw whunges tnc lhse furnestyre wetsi lw ptades gtfe le
Iter 100: en lw waunged rni lade futnedrbte werdm lw yraced grfe le ma
Iter 200: en lw wrunges ani lrse futnesabte weasm lw yaices gafe le mr
Iter 300: en lw wrunges ani lrse futnesabte weasm lw yaices gafe le mr
Iter 400: en lo oounges anf lose iuinesabie oeasr lo yattes gaie le ro
Iter 500: en mo oounget anf mote iuinetabie oeatr mo yastet gaie me ro
Iter 600: on mf founeet ang mote iuinetabie featr mf fastet eaie me ro
Iter 700: on mf founeet ang mote iuinetabie featr mf fastet eaie me ro
Iter 800: on mf founeet ang mote iulnetable featr mf fastet eaie me ro
Iter 900: on mf founeet ang mote iulnetaple featr mf fastet eaie me ro
Iter 1000: on mf founeet ang mote hulnetaple featr mf fastet eahe me ro
Iter 1100: on mf founeet ang mote hulnetaple featr mf fastet eahe me ro
Iter 1200: on mf founeet ang mote hulnetaple featr mf fastet eahe me ro
Iter 1300: on mf founeet ang mote wulnetaple featr mf fastet eawe me ro
Iter 1400: on mf founeet ang mote wulnetaple featr mf fastet eawe me ro
Iter 1500: on mf founees ang mose wulnesaple feasr mf fattes eawe me ro
Iter 1600: on mf fountes ang mose iulnesaple feasr mf fathes taie me ro
Iter 1700: in mf fountes and mose kulnesaple feasr mf fathes take me ro
Iter 1800: in mf fountes and mose kulnesaple feasr mf fathes take me ro
Iter 1900: in my yountes and mose bulnesaple yeasr my fathes tabe me ro
Iter 2000: in my yountes and mose kulnesaple yeasr my fathes take me ro
Iter 2100: in my yountes and mose kulnesaple yeasr my fathes take me ro
Iter 2200: in my younter and more kulneraple years my father take me so
Iter 2300: in my younter and more kulneraple years my father take me so
Iter 2400: in me eounder and more kulneraple eears me father dake me so
Iter 2500: in me eounder and more kulneraple eears me father dake me so
Iter 2600: in me eounder and more kulneraple eears me father dake me so
Iter 2700: in me eounder and more vulneraple eears me father dave me so
Iter 2800: in me eounder and more vulneraple eears me father dave me so
Iter 2900: in me eounder and more vulneraple eears me father dave me so
Iter 3000: in me eounder and more vulneraple eears me father dave me so
Iter 3100: in me eounder and more vulneraple eears me father dave me so
Iter 3200: in me eounder and more vulnerafle eears me father dave me so
Iter 3300: in me eounder and more vulneraple eears me father dave me so
Iter 3400: in me eounder and more vulnerable eears me father dave me so
Iter 3500: in me eounder and more vulnerable eears me father dave me so
Iter 3600: in me eounder and more vulnerable eears me father dave me so
Iter 3700: in me eounder and more vulnerable eears me father dave me so
Iter 3800: in me eounder and more vulnerable eears me father dave me so
Iter 3900: in me eounder and more vulnerable eears me father dave me so
Iter 4000: in me eounder and more vulnerable eears me father dave me so
Iter 4100: in my younder and more vulnerable years my father dave me so
Iter 4200: in my younder and more vulnerable years my father dave me so
Iter 4300: in my younder and more vulnerable years my father dave me so

Iter 4400: in my younder and more vulnerable years my father dave me so
Iter 4500: in my younder and more vulnerable years my father dave me so
Iter 4600: in my younder and more vulnerable years my father dave me so
Iter 4700: in my younder and more vulnerable years my father dave me so
Iter 4800: in my younder and more vulnerable years my father dave me so
Iter 4900: in my younder and more vulnerable years my father dave me so
Iter 5000: in my younder and more vulnerable years my father dave me so
Iter 5100: in my younder and more vulnerable years my father dave me so
Iter 5200: in my younder and more vulnerable years my father dave me so
Iter 5300: in my younder and more vulnerable years my father dave me so
Iter 5400: in my younder and more vulnerable years my father dave me so
Iter 5500: in my younder and more vulnerable years my father dave me so
Iter 5600: in my younder and more vulnerable years my father dave me so
Iter 5700: in my younder and more vulnerable years my father dave me so
Iter 5800: in my younder and more vulnerable years my father dave me so
Iter 5900: in my younder and more vulnerable years my father dave me so
Iter 6000: in my younder and more vulnerable years my father dave me so
Iter 6100: in my younder and more vulnerable years my father dave me so
Iter 6200: in my younder and more vulnerable years my father dave me so
Iter 6300: in my younder and more vulnerable years my father dave me so
Iter 6400: in my younder and more vulnerable years my father dave me so
Iter 6500: in my younder and more vulnerable years my father dave me so
Iter 6600: in my younder and more vulnerable years my father dave me so
Iter 6700: in my younder and more vulnerable years my father dave me so
Iter 6800: in my younder and more vulnerable years my father dave me so
Iter 6900: in my younder and more vulnerable years my father dave me so
Iter 7000: in my younder and more vulnerable years my father dave me so
Iter 7100: in my younder and more vulnerable years my father dave me so
Iter 7200: in my younder and more vulnerable years my father dave me so
Iter 7300: in my younder and more vulnerable years my father dave me so
Iter 7400: in my younder and more vulnerable years my father dave me so

Final decrypted text after 7500 iterations:

"in my younder and more vulnerable years my father dave me some advice that iave been turnind over in my mind ever since, gwhenever you feel lite criticifind any oneeg he told mee ghust remember that all the people in this world havenat had the advantades that youave had,g he didnat say any more but weave always been unusually communicative in a reserved waye and i understood that he meant a dreat deal more than that, in consequence iam inclined to reserve all huddmentse a habit that has opened up many curious natures to me and also made me the victim of not a few veteran bores, the abnormal mind is quict to detect and attach itself to this quality when it appears in a normal persone and so it came about that in collede i was unhustly accused of beind a politiciane because i was privy to the secret driefs of wilde untnown men, most of the confidences were unsoudhtoofrequently i have feidned sleepe preoccupatione or

a hostile levity when i realifed by some unmistatable sidn that an intimate revelation was quiverind on the horifonoofor the intimate revelations of yound men or at least the terms in which they express them are usually pladiaristic and marred by obvious suppressions, reservind huddments is a matter of infinite hope, i am still a little afraid of missind somethind if i fordet thate as my father snobbishly suddestede and i snobbishly repeat a sense of the fundamental decencies is parcelled out unequally at birth,"

We can see that the MCMC converges pretty quickly using our intitialization.

4.5

Was answered in the previous section 4.4.

4.6

Sampling using only symbol probabilities:

Utilizing solely symbol probabilities for decryption, as opposed to considering transitions, is generally inadequate. This method presumes that the English language can be accurately modeled by mere symbol frequency, implying that adjacent symbols in words are independent. However, English, like any other language, exhibits significant sequential dependencies between symbols and words. For example, the symbol pair "vt" is virtually non-existent in English vocabulary, though it might appear in other languages like Russian. This highlights another important aspect: decryption models are typically language-specific due to these varying sequential dependencies.

Relying exclusively on symbol probabilities would likely yield suboptimal decryption results. Even with access to the true stationary distribution of symbols in English, this approach might only be somewhat effective if the size of the encrypted text were extremely large, thereby closely mirroring the true distribution. However, in our case, the encrypted text is too small for this method to be effective, and furthermore, we do not have access to the actual stationary distribution of English.

Output :

Iter 0: on lw whunges tni lhse furnestyre wetsi lw ptades gtfe le ih
Iter 100: en dc counges tni dose aurnestwre cetsi dc ftahes gtae de io
Iter 200: ea ol louages tai oose nuraestwre letsi ol ctahes gtne oe io
Iter 300: ea id dtuages tai itse nuraestwre detso id ltaoes gtne ie ot
Iter 400: ta in ntwameh eai ithe dwraeheure neeho in leaoeh mede ie ot
Iter 500: ta hn ntcadeo aai htoe dcraeoalre neaoo hn uaeieo dade he ot
Iter 600: ta hn nacadeo tai haoe dchaeotuhe netoo hn lteieo dtde he oa

Iter 700: ta hn nacareo tao haoe dchaeotfhe netoi hn lteieo rtde he ia
Iter 800: ta hn nasadeo tao haoe rshaeotfhe netoi hn wteieo dtre he ia
Iter 900: ta hh hisaleo tao hioe dsiaeotfie hetoa hh uteneo ltde he ai
Iter 1000: ta hs sihaleo tao hioe dhiaeotwie setoa hs uteneo ltde he ai
Iter 1100: ta hs sihareo tao hioe dhiaeotwie setoa hs uteneo rtde he ai
Iter 1200: ta hs sohareo tai hooe dhiaeotfie setoa hs uteneo rtde he ao
Iter 1300: tt hs sontdeo ati hooe rniteoafie seaoa hs uaeheo dare he ao
Iter 1400: tt hr rnotdeo ati hnoe soiteoafie reaoa hr caeheo dase he an
Iter 1500: tt hr rnitdeo ati hnoe sioteoawoe reaoa hr caeheo dase he an
Iter 1600: tt hr rhitdeo ati hhoe sioteoawoe reaoa hr maeneo dase he ah
Iter 1700: tt hm miitdeo ath hioe rioteoadoe meaoa hm laeneo dare he ai
Iter 1800: tt hm miitreo atn hioe dioteoadoe meaoa hm laeheo rade he ai
Iter 1900: tt hm miitreo atn hioe dioteoadoe meaoa hm laeheo rade he ai
Iter 2000: tt hm miitreo atn hioe dioteoadoe meaoa hm laeheo rade he ai
Iter 2100: tt hl liitreo atn hioe dioteoadoe leaoa hl maeheo rade he ai
Iter 2200: at od dhitreh ttn ohhe diotehtloe detha od mteseh rtde oe ah
Iter 2300: ta ni ihiameh tao nhhe dioaehtloe ietha ni rteseh mtde ne ah
Iter 2400: ta ni ihoameh tao nhhe doiaehtlie ietha ni rteseh mtde ne ah
Iter 2500: ta nd dhsameh tao nhhe dsiaehtlie detha nd rteoeh mtde ne ah
Iter 2600: ta nd dhsameh tai nhhe dsiaehtlie detha nd rteoeh mtde ne ah
Iter 2700: ta od dhsameh tai ohhe dsiaehtlie detha od rteneh mtde oe ah
Iter 2800: ta od dhsameh tai ohhe dsiaehtlie detha od rteneh mtde oe ah
Iter 2900: ta od dhsameo tai ohoe dsiaeotlie detoa od rteneo mtde oe ah
Iter 3000: ta nd dhialeo tao nhoe diiaeotmie detoa nd rteseo ltde ne ah
Iter 3100: ta nl looadeh tai nohe doiaehtmie letha nl rteseh dtde ne ao
Iter 3200: ta nl looadeh tai nohe doiaehtmie letha nl rteseh dtde ne ao
Iter 3300: ta nl looareh tai nohe doiaehtmie letha nl dteseh rtde ne ao
Iter 3400: ta ol lonareh tai oohe dniaehtmie letha ol dteseh rtde oe ao
Iter 3500: ta hr ranaleo tai haoe dniaeotmie retoo hr dteseo ltde he oa
Iter 3600: ta ih hanaleo tah iaoe dniaeotmie hetoo ih dteseo ltde ie oa
Iter 3700: ta ih haialeo tah iaoe dinaeotwne hetoo ih dteseo ltde ie oa
Iter 3800: ta ih haiadeo tah iaoe linaeotgne hetoo ih dteseo dtle ie oa
Iter 3900: ta ir rohadeo tah iooe lhnaeotgne retoa ir dteseo dtle ie ao
Iter 4000: ta ir rosadeo tah iooe lsnaeotgne retoa ir dteheo dtle ie ao
Iter 4100: at ir rostdeo tth iooe lsnteotgne retoa ir dteheo dtle ie ao
Iter 4200: at ir rostdeo ttn iooe dshteotghe retoa ir lteheo dtde ie ao
Iter 4300: at ir rastdeo tth iaoe dsoteotgoe reton ir lteheo dtde ie na
Iter 4400: at hr rastdeo tti haoe dsnteotmne retoo hr lteieo dtde he oa
Iter 4500: at hr rastdeo tti haoe lsnteotmne retoo hr hteieo dtle he oa
Iter 4600: at hr rastdeo tti haoe lsnteotmne retoo hr hteieo dtle he oa
Iter 4700: at hr rasthen tti hane lsotentmoe retno hr dteien htle he oa
Iter 4800: at hr risthea ttn hiae lsoteatuoe retao hr dteiea htle he oi
Iter 4900: at hr rosthea ttn hoae lsiteatmie retai hr dteoea htle he io
Iter 5000: at hr rosthea ttn hoae lsiteatmie retai hr dteoea htle he io
Iter 5100: at ir rosthea ttn ioae lshteatmhe retai ir dteoea htle ie io
Iter 5200: at ih hostrea ttn ioae lshteatmhe hetai ih dteoea rtle ie io

Iter 5300: at ih hastren tti iane lshtentuhe hetno ih dteoen rtle ie oa
Iter 5400: at oh hastren tti oane lshtentuhe hetno oh dteien rtle oe oa
Iter 5500: at oh hostren ttd oone lshtentuhe hetna oh dteien rtle oe ao
Iter 5600: at oh hostdei ttd ooie lshteituhe hetia oh rtenei dtle oe ao
Iter 5700: at sh hiotdei ttd siie lohteituhe hetia sh rtenei dtle se ai
Iter 5800: at nh hiotdei eto niie lohteieuhe heeia nh detsei dele ne ai
Iter 5900: at nh hiotdei ets niie lohteieuhe heeia nh detoei dele ne ai
Iter 6000: at ih hnotdei ets inie looteieuoe heeia ih dethei dele ie an
Iter 6100: at ih hnotdei ets inie looteiefoe heeia ih dethei dele ie an
Iter 6200: at ih hootdei ets ioie lonteiefne heeia ih dethei dele ie ao
Iter 6300: at ih hootdei ets ioie lonteieune heeia ih dethei dele ie ao
Iter 6400: at ih hootdei ets ioie fonteieune heeia ih dethei defe ie ao
Iter 6500: at ih hootdei ets ioie fonteieune heeia ih dethei defe ie ao
Iter 6600: ta ih hooadei eas ioie fonaeieune heeia ih dethei defe ie ao
Iter 6700: ta ih hooadei eas ioie fonaeiedne heeia ih uethei defe ie ao
Iter 6800: ta ih hooadei eas ioie fonaeiedne heeia ih uethei defe ie ao
Iter 6900: ta ih hooadei eas ioie fohaeiedhe heeia ih uetnei defe ie ao
Iter 7000: ta hs sooadeh eai hohe foiaehedie seeha hs uetneh defe he ao
Iter 7100: ta hs sooadeh eai hohe uoiaehedie seeha hs fetneh deue he ao
Iter 7200: ta ir ronadeh eai iohe dnhaehefhe reeha ir uetoeh dede ie ao
Iter 7300: aa ir ronadei eah ioie dnhaeiefhe reeit ir uetoei dede ie to
Iter 7400: aa ir ronadei eah ioie lnhaeiefhe reeit ir uetoei dele ie to

Final decrypted text:

"aa ir rtoadei tah itie dohaeitfhe retin ir uteoei dtde ie ntie thdase eote a,de feea eoiaaad tdei aa ir iaah edei naasef mcoeaedei rto ueeh habe siaeasakaad tar taewm oe ethh iew myone ieieifei eote thh eoe letlhe aa eoan ctihh otdea,e oth eoe thdtaetden eote rto,de othfm oe haha,e ntr tar itie foe ce,de thctrn feea oaonothhr stiioaasteade aa t ieneideh ctrw tah a oaheinetth eote oe ietae t diete heth itie eota eotef aa staneyoease a,i aashaaeh et ieneide thh yohdieaenw t otfae eote otn tleaeh ol itar soiaton ateoien et ie tah thnt ithe ie eoe daseai tu ate t uec deeeita ftienf eoe tfatiith iaah an yoasb et heeese tah teetso aenehu et eoan yothaer coea ae tlletin aa t atiith leintaw tah nt ae stie tftoe eote aa sthhede a ctn oayonehr tssoneh tu feaad t lthaeasataw festone a ctn liadr et eoe nesiee diaeun tu cahhw oabatca ieaf itne tu eoe stauaheasen ceie oantodoegguieyoeaehr a otde ueadaeh nheelw lietssolteataw ti t otneahe hedaer coea a iethakeh fr ntie oaianetbtfhe nada eote ta aaeaitee iedehteata ctn yoadeiaad ta eoe otiaktagguti eoe aaeaitee iedehteatan tu rtoad iea ti te hetne eoe eeiin aa coaso eoer eqlienn eoei tie onothhr lhtdatianeas tah itiieh fr tfdaton nolliennatanf ieneidaad yohdieaen an t iteeei tu aauaaaee otlef a ti neahh t haeehe tuitah tu iannaad ntieeoaad au a utidee eotew tn ir uteoei natffanohr noddeneehw tah a natffanohr ielete t neane tu eoe uoahtieaeth heseasaen an ltisehheh toe oaeyothhr te faieof"

As you can see the decrypted text just contains the most likely English symbols, which are mostly vowels and spaces.

Using a second-order Markov chain:

Using a second-order Markov chain for English text decryption theoretically enhances accuracy by capturing the strong sequential dependencies of symbols within words, offering a more nuanced approach to decryption. However, this theoretical advantage doesn't necessarily translate into practical superiority. The increased accuracy comes at a cost, primarily due to the substantial rise in model complexity, which may not justify the marginal gains in efficiency. This complexity arises from the quadrupling of possible states, resulting in an exponentially larger transition matrix.

Implementing such a model would demand significantly more computational resources and a more extensive "training text" than 'War and Peace' to accurately estimate empirical frequencies of symbol triplets. For our purposes, with 'War and Peace' and the 'message.txt', we would encounter an even greater number of zero empirical triplets compared to zero empirical transition probabilities in our current model. This higher sparsity rate indicates that the number of triplets present in 'War and Peace' will be insufficient for reliable transition probability estimation. We would have to artificially increase the probability of these transitions to retain ergodicity, leading to an even more unreliable transition matrix. Therefore, the trade-off between results and increased complexity suggests that a first-order model might be a more balanced choice in terms of efficiency and outcome in our context.

One to two mapping scheme:

If an encryption scheme maps two different symbols to the same encrypted value, utilizing permutations for decryption becomes impractical due to the loss of one-to-one mapping. The effectiveness of permutations in our previous approach was anchored on the assumption that the encryption function was bijective, allowing us to decrypt 'message.txt' using proposed and current permutations (which are inverse functions of potential encryption functions) and then evaluate the likelihood for comparison. This inverse mapping was deterministic. With a non-unique mapping, however, the inverse becomes stochastic, necessitating the implementation of a Hidden Markov Model (HMM). In this model, the potential decrypted symbols become latent variables, each having two possible states.

While the transition matrix and initial stationary distributions remain the same, the introduction of emission probabilities (which represent the likelihood of each encrypted symbol corresponding to each possible decrypted symbol) adds complexity, as these probabilities are unknown. We lack insight into the encryption process of 'message.txt', meaning an Expectation-Maximization (EM) algorithm like Baum-Welch would be required to learn these probabilities, from the data. This added complexity arises from not assuming a one-to-one mapping, and EM algorithms can be computationally intensive, particularly with a model comprising 53 symbols. The risk of converging to local optima is high, especially if

initial parameters are not optimally chosen. Therefore, assuming a non-unique encryption scheme is not the most effective approach for decryption, unless the text was originally encrypted using a stochastic method.

Case of Chinese language:

For the Chinese language, the challenge escalates dramatically due to the immense size of the transition matrix, potentially encompassing more than 100 million possible transitions. Even discounting the use of this model for decryption, a substantially larger corpus of text would be necessary to accurately estimate the transition probabilities, rendering the task computationally unfeasible. Thus, it is not a viable approach for decrypting Chinese text.

5 Optimization

5.1

(We will use the same format given in the slides here). We apply the method of Lagrange multipliers to find the local extrema of the function $f(x, y) = x + 2y$ subject to the constraint $y^2 + xy = 1$. We introduce the Lagrange multiplier λ and set up the following system of equations:

$$\begin{cases} \nabla f(x, y) + \lambda \nabla g(x, y) = 0 \\ g(x, y) = y^2 + xy - 1 = 0 \end{cases}$$

where $g(x, y)$ is the constraint function. The gradients of f and g are:

$$\nabla f(x, y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) = (1, 2)$$
$$\nabla g(x, y) = \left(\frac{\partial g}{\partial x}, \frac{\partial g}{\partial y} \right) = (y, 2y + x)$$

Substituting these into the system, we get:

$$1 + \lambda y = 0 \tag{1}$$
$$2 + \lambda(2y + x) = 0 \tag{2}$$
$$y^2 + xy - 1 = 0 \tag{3}$$

Note that we assume $\lambda \neq 0$ for the solution to be meaningful in the context of the Lagrange multiplier method. Now, taking $(2) - 2(1)$ gives $\lambda x = 0 \Rightarrow x = 0$ and so $\lambda \neq 0$ is arbitrary.

$\Rightarrow$ (3) becomes $y^2 - 1 = 0 \Rightarrow y = -1$ or $y = 1$

Local minima is at $(x, y) = (0, 1)$ as $\nabla f = \nabla g$ i.e gradients point in the same direction at $(0, 1)$.

Local maxima is at $(x, y) = (0, -1)$ as $\nabla f = -\nabla g$ i.e gradients point in opposite directions at $(0, -1)$.

5.2

To compute $\ln(a)$ for a given $a \in \mathbb{R}^+$ using Newton's method, we first need to derive a suitable function $f(x, a)$ such that finding the root of $f(x, a) = 0$ will yield $x = \ln(a)$. The function can be defined as:

$$f(x, a) = e^x - a$$

The root of this function occurs when $e^x = a$, which implies $x = \ln(a)$.

Newton's method updates the estimate of the root iteratively using the formula:

$$x_{n+1} = x_n - \frac{f(x_n, a)}{f'(x_n, a)}$$

For our function $f(x, a) = e^x - a$, the derivative $f'(x, a)$ with respect to x is e^x. Thus, the update equation becomes:

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

$$\Rightarrow x_{n+1} = x_n + ae^{-x_n} - 1$$

Which will be the update used iteratively to approximate $\ln(a)$.

6 Eigenvalues as solutions of an optimization problem.

Let A be a symmetric $n \times n$-matrix, and define

$$q_A(x) := x^T Ax \quad \text{and} \quad R_A(x) := \frac{x^T Ax}{x^T x} = \frac{q_A(x)}{\|x\|^2}$$

for $x \in \mathbb{R}^n \setminus \{\mathbf{0}\}$, as if x is the zero vector then $R_A(x)$ is undefined. So from now on we will always assume that x is non-zero.

6.1

As we know, an eigenvector is unique up to scaling. It's the eigendirection that we are interested in, not the magnitude of an eigenvector. Thus, we will show that the constraint $\|x\| = 1$ doesn't change the argmax of $R_A(x)$. Let c be any non-zero real scalar.

$$R_A(cx) = \frac{(cx^T)A(cx)}{\|cx\|^2} = \frac{c^2}{c^2}\frac{x^TAx}{\|x\|^2} = R_A(x)$$

$$\Rightarrow R_A(cx) = R_A(x) \quad \forall c \in \mathbb{R}, \quad c \neq 0.$$

Thus, $\hat{x} = \underset{x\in\mathbb{R}^n\setminus\{\mathbf{0}\}}{\operatorname{argmax}} R_A(x) = \underset{\substack{x\in\mathbb{R}^n\setminus\{\mathbf{0}\}\\ \|x\|^2=1}}{\operatorname{argmax}} R_A(x)$

So, $R_A(x)$ can be defined on compact set $S = \{x \in \mathbb{R}^n \setminus \{\mathbf{0}\} \mid \|x\| = 1\}$ and as $q_A(x)$ is continuous on $\mathbb{R}$ then $\frac{q_A(x)}{\|x\|} = q_A(x) = R_A(x)$ is continuous on S. Thus, by the extreme value theorem of calculus $R_A(x)$ is bounded and $\exists v, w \in S$ such that $R_A(v) = \sup_{x\in S} R_A(x), \quad R_A(w) = \inf_{x\in S} R_A(x)$.

6.2

Let $\lambda_1 \geq \ldots \geq \lambda_n$ be the eigenvalues of A enumerated by decreasing size, and $v_1, \ldots, v_n$ be the corresponding eigenvectors that form an ONB.

$$Ax = A\sum_{i=1}^{n}(v_i^Tx)v_i = \sum_{i=1}^{n}(v_i^Tx)\lambda_iv_i \leq \lambda_1\sum_{i=1}^{n}(v_i^Tx)v_i = \lambda_1x$$

As $\lambda_1 \geq \ldots \geq \lambda_n$, and $(v_i^Tx) \in \mathbb{R} \Rightarrow A(v_i^Tx)v_i = (v_i^Tx)Av_i = (v_i^Tx)\lambda_iv_i$.

$$\Rightarrow Ax \leq \lambda_1x$$

$$\Leftrightarrow x^TAx \leq \lambda_1\|x\|^2$$

$$\Rightarrow R_A(x) \leq \lambda_1 \quad \forall x \in \mathbb{R}^n \setminus \{\mathbf{0}\}$$

6.3

Assume $x \in \mathbb{R}^n \setminus \{\mathbf{0}\}, \quad x \notin \text{span}\{v_1, \ldots, v_k\}$. Then,

$$R_A(x) = \frac{(\sum_{i=1}^n (v_i^T x)v_i)^T A(\sum_{i=1}^n (v_i^T x)v_i)}{\|x\|^2}$$

$$= \frac{(\sum_{i=1}^n (v_i^T x)v_i)^T (\sum_{i=1}^n (v_i^T x)\lambda_i v_i)}{\|x\|^2}$$

$$= \frac{(\sum_{i=1}^k (v_i^T x)v_i + \sum_{j=k+1}^n (v_j^T x)v_j)^T (\sum_{i=1}^k (v_i^T x)\lambda_1 v_i + \sum_{j=k+1}^n (v_j^T x)\lambda_j v_j)}{\|x\|^2}$$

$$\Rightarrow R_A(x) = \frac{\lambda_1 \sum_{i=1}^k (v_i^T x)^2 + \sum_{j=k+1}^n \lambda_j (v_j^T x)^2}{\|x\|^2}$$

We have $\|x\|^2 = (\sum_{i=1}^n (v_i^T x)v_i)^T (\sum_{i=1}^n (v_i^T x)v_i) = \sum_{i=1}^n (v_i^T x)^2 = \sum_{i=1}^k (v_i^T x)^2 + \sum_{j=k+1}^n (v_j^T x)^2$ and using results from previous question we get that,

$$R_A(x) = \frac{\lambda_1 \sum_{i=1}^k (v_i^T x)^2 + \sum_{j=k+1}^n \lambda_j (v_j^T x)^2}{\|x\|^2} \leq \lambda_1$$

$$\Leftrightarrow \frac{\sum_{i=1}^k (v_i^T x)^2 + \sum_{j=k+1}^n (\frac{\lambda_j}{\lambda_1})(v_j^T x)^2}{\sum_{i=1}^k (v_i^T x)^2 + \sum_{j=k+1}^n (v_j^T x)^2}$$

Here, equality holds if :

$\lambda_j = \lambda_1 \quad \forall j \geq k+1 \Leftrightarrow k = n$ but this contradicts our assumption that $x \notin \text{span}\{v_1, \ldots, v_k\}$.

Or, if $(v_j^T x) = 0 \quad \forall j \geq k+1 \Rightarrow v_i^T x \neq 0 \quad \forall i \in [k]$ (as x is a non-zero real vector), which would imply that $x = \sum_{i=1}^k (v_i^T x)v_i \quad \Leftrightarrow \quad x \in \text{span}\{v_1, \ldots, v_k\}$ which is again a contradiction to our initial assumption.

Thus, by contradiction, $x \notin \text{span}\{v_1, \ldots, v_k\} \Rightarrow R_A(x) < \lambda_1$.

A Appendix: Code for MCMC

This is the code before applying the MCMC sampling, i.e how we treat the problem:

```
import numpy as np
import random
from collections import Counter
import pandas as pd
from scipy.special import logsumexp
import matplotlib.pyplot as plt
import numpy as np

# Step 1: Read the list of symbols
with open('symbols.txt', 'r') as file:
    symbols = set(file.read().splitlines())
    symbols.add(' ')

# Define a function to check if a word contains only valid symbols (excluding specific symbols and numbers)
def is_valid_word(word, valid_symbols):
    return all(char in valid_symbols for char in word)
# Step 2: Read the entire War_filtered_manually.txt file into a single string. T
with open('War_filtered_mann.txt', 'r', encoding='utf-8') as file:
    text = file.read()

# Replace non-standard quotes with standard ones and remove newlines
text = text.replace('\n', ' ').replace('\r', ' ')

# Step 3: Convert to lowercase
text = text.lower()

# Step 4: Process the text
# We'll use a list comprehension to filter out invalid words as the text contains a lot of Russian names
valid_words = [word for word in text.split() if is_valid_word(word, symbols)]

# Join the valid words back into a single string
cleaned_text = ' '.join(valid_words)

# Now, let's write the cleaned text to a new file as one continuous line
with open('Cleaned_War.txt', 'w', encoding='utf-8') as file:
    file.write(cleaned_text)

from collections import defaultdict
synthetic_count = 0
# Initialize count dictionaries
symbol_counts = defaultdict(int, {symbol: 0 for symbol in symbols})
pair_counts = defaultdict(int, {(prev_symbol, next_symbol): synthetic_count for prev_symbol in symbols for next_symbol in symbols})
```

```
43
44 # Count symbols and pairs
45 prev_symbol = None
46 for symbol in cleaned_text:
47     symbol_counts[symbol] += 1
48     if prev_symbol is not None:
49         pair_counts[(prev_symbol, symbol)] += 1
50     prev_symbol = symbol
51
52 # Total number of symbols
53 total_symbols = sum(symbol_counts.values())
54
55 # Calculate the stationary distribution   (  )
56 phi = {symbol: count / total_symbols for symbol, count in
       symbol_counts.items()}
57
58 psi = {(prev_symbol, next_symbol): 0 for prev_symbol in symbols for
        next_symbol in symbols}
59
60 for prev_symbol in symbols:
61     # This sum will be the total count of all transitions from 'a'
       to any 'x'
62     total_transitions = sum(pair_counts[(prev_symbol, next_symbol)]
        for next_symbol in symbols)
63
64     if total_transitions == 0:
65         # If there are no transitions from prev_symbol, assign 0 to
        all its transition probabilities
66         for next_symbol in symbols:
67             psi[(prev_symbol, next_symbol)] = 0
68     else:
69         # Now calculate the probability for each 'a' to 'x'
       transition
70         for next_symbol in symbols:
71             count_s_to_x = pair_counts[(prev_symbol, next_symbol)]
72             psi[(prev_symbol, next_symbol)] = count_s_to_x /
       total_transitions
73
74
75 #Note that due to the context of the text we have we have a lot of
       dialogue so the " has more importance than it should be
76
77 # Create a transition matrix psi with zeros for all possible
       transitions
78 transition_matrix = np.zeros((len(symbols), len(symbols)))
79
80 # Create a mapping of symbol to index to keep track of the matrix
       indices
81 symbol_to_index = {symbol: idx for idx, symbol in enumerate(symbols
       )}
82 index_to_symbol = {i: symbol for i, symbol in enumerate(symbols)}
83
84 for (source, destination), probability in psi.items():
85     row_idx = symbol_to_index[source]
86     col_idx = symbol_to_index[destination]
87     transition_matrix[row_idx, col_idx] = probability
88
```

```
# Apply log to transition matrix and initial probabilities and add small constant to 0 probabilities or probabilities that might cause instabilities in the log domain (to small).
transition_matrix = np.where(transition_matrix == 0, np.exp(-18), transition_matrix)
log_transition_matrix = np.log(transition_matrix)
log_transition_matrix = np.where(log_transition_matrix > -18, log_transition_matrix, -18)
log_phi = {symbol: np.log(max(prob, np.exp(-18))) for symbol, prob in phi.items()}
```

This is the code for the MH algorithm:

```
def initialize_permutation_by_frequency(encrypted_text, symbols, phi):
    # Count the frequency of each symbol in the encrypted text
    encrypted_freq = Counter(encrypted_text)

    # Sort symbols by frequency in the encrypted text
    sorted_encrypted_symbols = [item[0] for item in encrypted_freq.most_common()]

    # Sort symbols by frequency in War and Peace
    sorted_symbol_freq = [symbol for symbol, _ in sorted(phi.items(), key=lambda item: item[1], reverse=True)]

    # Creating an initial permutation that maps the most frequent encrypted symbol to the most frequent symbol in War and Peace.
    permutation_mapping = dict(zip(sorted_encrypted_symbols, sorted_symbol_freq))
    return [permutation_mapping.get(symbol, symbol) for symbol in symbols]

def decrypt(encrypted_text, symbol_to_index, permutation):
    # This function will decrypt the text based on the current permutation
    return ''.join(permutation[symbol_to_index[symbol]] for symbol in encrypted_text)

def calculate_log_likelihood(decrypted_text, log_phi, log_transition_matrix, symbol_to_index):
    # This function will calculate the log-likelihood of the decrypted text
    # based on the stationary distribution and transition probabilities
    log_likelihood = log_phi[decrypted_text[0]]  # Start with the stationary probability of the first symbol
    for i in range(1, len(decrypted_text)):
        current_symbol = decrypted_text[i-1]
        next_symbol = decrypted_text[i]
        log_likelihood += log_transition_matrix[symbol_to_index[current_symbol], symbol_to_index[next_symbol]]
    return log_likelihood

def metropolis_hastings(encrypted_text, symbol_to_index, index_to_symbol, log_phi, log_transition_matrix, num_iterations
```

```
    =7500):
    # This function will implement the Metropolis-Hastings
    algorithm
    current_permutation = initialize_permutation_by_frequency(
    encrypted_text, symbols, phi)
    current_decrypted_text = decrypt(encrypted_text,
    symbol_to_index, current_permutation)
    current_log_likelihood = calculate_log_likelihood(
    current_decrypted_text, log_phi, log_transition_matrix,
    symbol_to_index)

    for iteration in range(num_iterations):
        # Propose a new permutation
        proposed_permutation = current_permutation.copy()
        i, j = np.random.choice(len(symbols), 2, replace=False)
        proposed_permutation[i], proposed_permutation[j] =
    proposed_permutation[j], proposed_permutation[i]

        # Decrypt text with the proposed permutation and calculate
    its log-likelihood
        proposed_decrypted_text = decrypt(encrypted_text,
    symbol_to_index, proposed_permutation)
        proposed_log_likelihood = calculate_log_likelihood(
    proposed_decrypted_text, log_phi, log_transition_matrix,
    symbol_to_index)

        # Calculate the change in log-likelihood
        delta_log_likelihood = proposed_log_likelihood -
    current_log_likelihood

        # Decide whether to accept the new permutation
        if delta_log_likelihood > 0 or np.log(np.random.rand()) <
    delta_log_likelihood:
            current_permutation = proposed_permutation
            current_log_likelihood = proposed_log_likelihood

        # Report the current decryption of the first 60 symbols
    after every 100 iterations
        if iteration % 100 == 0:
            decrypted_sample = decrypt(encrypted_text[:60],
    symbol_to_index, current_permutation)
            print(f"Iter {iteration}: {decrypted_sample}")

    return decrypt(encrypted_text, symbol_to_index,
    current_permutation)


# Load the encrypted message
with open('message.txt', 'r') as file:
    encrypted_text = file.read().strip()

# Run the Metropolis-Hastings algorithm
decrypted_text = metropolis_hastings(encrypted_text,
    symbol_to_index, index_to_symbol, log_phi,
    log_transition_matrix)
print(decrypted_text)
```

This is the code for MH algorithm when only considering the initial probabilities:

```
def calculate_log_likelihood_from_stationary(proposed_decrypted_text, log_phi, symbol_to_index):
    # This function will calculate the log-likelihood of the decrypted text
    # based solely on the stationary distribution of symbol probabilities
    log_likelihood = 0
    for symbol in proposed_decrypted_text:
        log_likelihood += log_phi[symbol]
    return log_likelihood

def metropolis_hastings_stationary(encrypted_text, symbol_to_index, index_to_symbol, log_phi, num_iterations=7500):
    # This function will implement the Metropolis-Hastings algorithm
    current_permutation = initialize_permutation_by_frequency(encrypted_text, symbols, phi)
    current_decrypted_text = decrypt(encrypted_text, symbol_to_index, current_permutation)
    current_log_likelihood = calculate_log_likelihood_from_stationary(current_decrypted_text, log_phi, symbol_to_index)

    for iteration in range(num_iterations):
        # Propose a new permutation
        proposed_permutation = current_permutation.copy()
        i, j = np.random.choice(len(symbols), 2, replace=False)
        proposed_permutation[i], proposed_permutation[j] = proposed_permutation[j], proposed_permutation[i]

        # Decrypt text with the proposed permutation and calculate its log-likelihood
        proposed_decrypted_text = decrypt(encrypted_text, symbol_to_index, proposed_permutation)
        proposed_log_likelihood = calculate_log_likelihood_from_stationary(proposed_decrypted_text, log_phi, symbol_to_index)

        # Calculate the change in log-likelihood
        delta_log_likelihood = proposed_log_likelihood - current_log_likelihood

        # Decide whether to accept the new permutation
        if delta_log_likelihood > 0 or np.log(np.random.rand()) < delta_log_likelihood:
            current_permutation = proposed_permutation
            current_log_likelihood = proposed_log_likelihood

        # Report the current decryption of the first 60 symbols after every 100 iterations
        if iteration % 100 == 0:
            decrypted_sample = decrypt(encrypted_text[:60], symbol_to_index, current_permutation)
            print(f"Iter {iteration}: {decrypted_sample}")
```

```
37
38      return decrypt(encrypted_text, symbol_to_index,
        current_permutation)
39
40
41  # Load the encrypted message
42  with open('message.txt', 'r') as file:
43      encrypted_text = file.read().strip()
44
45  # Run the Metropolis-Hastings algorithm
46  decrypted_text = metropolis_hastings_stationary(encrypted_text,
        symbol_to_index, index_to_symbol, log_phi)
47  print(decrypted_text)
```