COMP0086 Summative Assignment

Nov 14, 2022

Question 1

(a) Our sample space for images is $\{0,1\}^D$, where each of our D dimensions can only take binary values (D being the number of pixels in the image). The exponential family best suited on this sample space is the D-dimensional multivariate Bernoulli distribution because it shares the same sample space. On the other hand, a D-dimensional multivariate Gaussian has the sample space $\mathbb{R}^D$, which does not match the sample space of our data. It is not immediately clear how the likelihood of an image of binary (discrete) values would be calculated under the continuous distribution of a multivariate Gaussian. Thus it would be inappropriate to model this dataset of images with a multivariate Gaussian.

(b) For $\{\mathbf{x}^{(n)}\}_{n=1}^N$, a data set of N images, the joint likelihood (assuming images are independently and identically distributed) is the product of N, D-dimensional multivariate Bernoulli distributions:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^N|\mathbf{p}) = \prod_{n=1}^N P(\mathbf{x}^{(n)}|\mathbf{p})$$

Substituting the D-dimensional multivariate Bernoulli:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^N|\mathbf{p}) = \prod_{n=1}^N \prod_{d=1}^D p_d^{x_d^{(n)}} (1-p_d)^{1-x_d^{(n)}}$$

Taking the logarithm, we get the log likelihood:

$$\mathcal{L}(\{\mathbf{x}^{(n)}\}_{n=1}^N|\mathbf{p}) = \sum_{n=1}^N \sum_{d=1}^D [x_d^{(n)} \log(p_d) + (1-x_d^{(n)}) \log(1-p_d)]$$

Note that since the logarithm is a monotonically increasing function on $\mathbb{R}_+$, the maximisers and minimisers of the likelihood do not change. Thus, to solve for the maximum likelihood estimate, $\hat{p}_d$, we can take the derivative of $\mathcal{L}(\{\mathbf{x}^{(n)}\}_{n=1}^N|\mathbf{p})$ with respect to p_d, the d^{th} element of $\mathbf{p}$:

$$\frac{\partial \mathcal{L}(\{\mathbf{x}^{(n)}\}_{n=1}^N|\mathbf{p})}{\partial p_d} = \sum_{n=1}^N \left(\frac{x_d^{(n)}}{p_d} - \frac{1-x_d^{(n)}}{1-p_d}\right)$$

$$\frac{\partial \mathcal{L}(\{\mathbf{x}^{(n)}\}_{n=1}^N|\mathbf{p})}{\partial p_d} = \frac{\sum_{n=1}^N x_d^{(n)}}{p_d} - \frac{\sum_{n=1}^N (1-x_d^{(n)})}{1-p_d}$$

and set the derivative to zero to solve for $\hat{p}_d$:

$$\frac{\sum_{n=1}^{N} x_d^{(n)}}{\hat{p}_d} - \frac{\sum_{n=1}^{N}(1-x_d^{(n)})}{1-\hat{p}_d} = 0$$

$$\sum_{n=1}^{N} x_d^{(n)} - \hat{p}_d \sum_{n=1}^{N} x_d^{(n)} - \hat{p}_d \cdot N + \hat{p}_d \sum_{n=1}^{N} x_d^{(n)} = 0$$

$$\hat{p}_d = \frac{1}{N}\sum_{n=1}^{N} x_d^{(n)}$$

Because we assume that each pixel is independent (we are taking the product of D one dimensional Bernoulli distributions), we can express the maximum likelihood for $\mathbf{p}$ in vectorised form as $\hat{\mathbf{p}}^{MLE}$:

$$\hat{\mathbf{p}}^{MLE} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}^{(n)}$$

(c) From Bayes' Theorem:

$$P(\mathbf{p}|\{\mathbf{x}^{(n)}\}_{n=1}^{N}) = \frac{P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|\mathbf{p})P(\mathbf{p})}{P(\{\mathbf{x}^{(n)}\}_{n=1}^{N})}$$

Taking the logarithm:

$$\mathcal{L}(\mathbf{p}|\{\mathbf{x}^{(n)}\}_{n=1}^{N}) = \mathcal{L}(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|\mathbf{p}) + \mathcal{L}(\mathbf{p}) - \mathcal{L}(\{\mathbf{x}^{(n)}\}_{n=1}^{N})$$

Taking the derivative with respect to p_d:

$$\frac{\partial \mathcal{L}(\mathbf{p}|\{\mathbf{x}^{(n)}\}_{n=1}^{N})}{\partial p_d} = \frac{\partial \mathcal{L}(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|\mathbf{p})}{\partial p_d} + \frac{\partial \mathcal{L}(\mathbf{p})}{\partial p_d}$$

where $\frac{\partial \mathcal{L}(\{\mathbf{X}^{(n)}\}_{n=1}^{N})}{\partial p_d}$=0 because it doesn't depend on p_d.
We know from (b):

$$\frac{\partial \mathcal{L}(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|\mathbf{p})}{\partial p_d} = \frac{\sum_{n=1}^{N} x_d^{(n)}}{p_d} - \frac{\sum_{n=1}^{N}(1-x_d^{(n)})}{1-p_d}$$

For the second term $\frac{\partial \mathcal{L}(\mathbf{p})}{\partial p_d}$, we start with $P(\mathbf{p})$, assuming each pixel to have an independent prior:

$$P(\mathbf{p}) = \prod_{d=1}^{D} P(p_d)$$

and assuming a Beta prior on each p_d:

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

Taking the logarithm:

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

Taking the derivative with respect to p_d:

$$\frac{\partial \mathcal{L}(\mathbf{p})}{\partial p_d} = \frac{(\alpha - 1)}{p_d} - \frac{(\beta - 1)}{1 - p_d}$$

Since we are only concerned with p_d, we are only left with a single element of the summation pertaining to p_d.

Combining, we have have an expression for $\frac{\partial \mathcal{L}(\mathbf{p}|\{\mathbf{x}^{(n)}\}_{n=1}^{N})}{\partial p_d}$:

$$\frac{\partial \mathcal{L}(\mathbf{p}|\{\mathbf{x}^{(n)}\}_{n=1}^{N})}{\partial p_d} = \frac{\sum_{n=1}^{N} x_d^{(n)}}{p_d} - \frac{\sum_{n=1}^{N}(1 - x_d^{(n)})}{1 - p_d} + \frac{(\alpha - 1)}{p_d} - \frac{(\beta - 1)}{1 - p_d}$$

$$\frac{\partial \mathcal{L}(\mathbf{p}|\{\mathbf{x}^{(n)}\}_{n=1}^{N})}{\partial p_d} = \frac{(\alpha - 1) + \sum_{n=1}^{N} x_d^{(n)}}{p_d} - \frac{(\beta - 1) + \sum_{n=1}^{N}(1 - x_d^{(n)})}{1 - p_d}$$

To find the maximum a posteriori (MAP) estimate $\hat{p_d}$ set $\frac{\partial \mathcal{L}(\mathbf{p}|\{\mathbf{x}^{(n)}\}_{n=1}^{N})}{\partial p_d} = 0$ and solve:

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

$$0 = (1 - \hat{p_d})(\alpha - 1) + (1 - \hat{p_d})\Big(\sum_{n=1}^{N} x_d^{(n)}\Big) - \hat{p_d}(\beta - 1) - \hat{p_d}\Big(\sum_{n=1}^{N}(1 - x_d^{(n)})\Big)$$

$$0 = (\alpha - \alpha\hat{p_d} + \hat{p_d} - 1) + \Big(\sum_{n=1}^{N} x_d^{(n)} - \hat{p_d}\sum_{n=1}^{N} x_d^{(n)}\Big) - (\hat{p_d}\beta - \hat{p_d}) - \Big(\hat{p_d} \cdot N - \hat{p_d}\sum_{n=1}^{N} x_d^{(n)}\Big)$$

Cancelling the $\hat{p_d}\sum_{n=1}^{N} x_d^{(n)}$ terms:

$$0 = \alpha - \alpha\hat{p_d} + \hat{p_d} - 1 + \sum_{n=1}^{N} x_d^{(n)} - \hat{p_d}\beta + \hat{p_d} - \hat{p_d} \cdot N$$

$$0 = \hat{p_d}(2 - \alpha - \beta - N) + \alpha - 1 + \sum_{n=1}^{N} x_d^{(n)}$$

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

Due to independence of our likelihood and priors for each dimension, we can express the maximum a priori for $\mathbf{p}$ in vectorised form as $\hat{\mathbf{p}}^{MAP}$:

$$\hat{\mathbf{p}}^{MAP} = \frac{\alpha - 1 + \sum_{n=1}^{N} \mathbf{x}^{n}}{(N + \alpha + \beta - 2)}$$

(d&e) The Python code for MLE and MAP:

```python
import matplotlib.pyplot as plt
import numpy as np


def _compute_maximum_likelihood_estimate(x: np.ndarray) -> np.ndarray:
    """
    Calculates MLE of images
    :param x: numpy array of shape (N, D)
    :return: MLE estimate
    """
    return np.mean(x, axis=0)


def _compute_maximum_a_priori_estimate(
    x: np.ndarray, alpha: float, beta: float
) -> np.ndarray:
    """
    Calculates MAP estimate of images
    :param x: numpy array of shape (N, D)
    :param alpha: param of prior distribution
    :param beta: param of prior distribution
    :return: MAP estimate
    """

    n, _ = x.shape
    return (alpha - 1 + np.sum(x, axis=0)) / (n + alpha + beta - 2)


def d(x: np.ndarray, figure_path: str, figure_title: str) -> None:
    """
    Produces answers for question 1d
    :param x: numpy array of shape (N, D)
    :param figure_path: path to store figure
    :param figure_title: figure title
    :return:
    """
    maximum_likelihood = _compute_maximum_likelihood_estimate(x)
    plt.figure()
    plt.imshow(
        np.reshape(maximum_likelihood, (8, 8)),
        interpolation="None",
    )
    plt.colorbar()
    plt.axis("off")
    plt.title(figure_title)
    plt.savefig(figure_path)


def e(
    x: np.ndarray, alpha: float, beta: float, figure_path: str, figure_title: str
) -> None:
    """
    Produces answers for question 1e
    :param x: numpy array of shape (N, D)
    :param alpha: param of prior distribution
    :param beta: param of prior distribution
    :param figure_path: path to store figure
    :param figure_title: figure title
    :return:
    """
    maximum_a_priori = _compute_maximum_a_priori_estimate(x, alpha, beta)
    plt.figure()
    plt.imshow(
        np.reshape(maximum_a_priori, (8, 8)),
        interpolation="None",
    )
    plt.colorbar()
    plt.axis("off")
    plt.title(figure_title)
    plt.savefig(f"{figure_path}.png")

    maximum_likelihood = _compute_maximum_likelihood_estimate(x)
    plt.figure()
    plt.imshow(
        np.reshape(maximum_a_priori - maximum_likelihood, (8, 8)),
        interpolation="None",
    )
    plt.colorbar()
    plt.axis("off")
    plt.title(f"MAP vs MLE")
    plt.savefig(f"{figure_path}-mle-vs-map.png")
```

src/solutions/q1.py

Displaying the learned parameters:

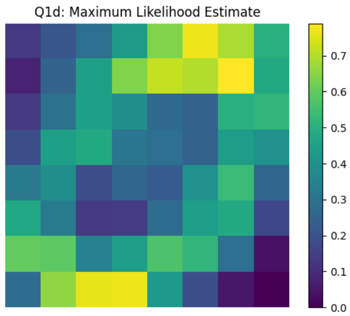

Figure 1: ML parameters

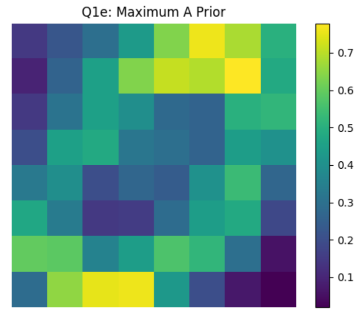

Figure 2: MAP parameters

Comparing the equations:

$$\hat{\mathbf{p}}^{MLE} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}^{(n)}$$

and

$$\hat{\mathbf{p}}^{MAP} = \frac{\alpha - 1 + \sum_{n=1}^{N}\mathbf{x}^{n}}{(N + \alpha + \beta - 2)}$$

As the number of data points increases, $\hat{\mathbf{p}}^{MAP}$ approaches $\frac{1}{N}\sum_{n=1}^{N}\mathbf{x}^{(n)}$, the $\hat{\mathbf{p}}^{MLE}$. This makes sense because as our data set gets bigger, the effect of the prior diminishes. However, if a specific pixel in all of the images of our data set are white or all black, the MLE for that pixel would either be 1 or 0. This may not be representative of our intuitions about images, as there should be some non-zero probability of a pixel being black or white. By introducing an appropriate prior we can ensure that the probability of that pixel will never be exactly zero or one. In our case, with a Beta(3,3) prior on each pixel, our parameter values are biased to be closer to 0.5 and to never be at the extremities 0 and 1. We can see this in Figure 2 where the range of our parameters is smaller than the range of Figure 1 and doesn't include zero. Figure 3 visualises $\hat{\mathbf{p}}^{MAP} - \hat{\mathbf{p}}^{MLE}$ and we can see that for likelihoods greater than 0.5 in the MLE, the MAP has a lower value and for likelihoods less than 0.5, the MAP has a higher value, confirming our intuitions.

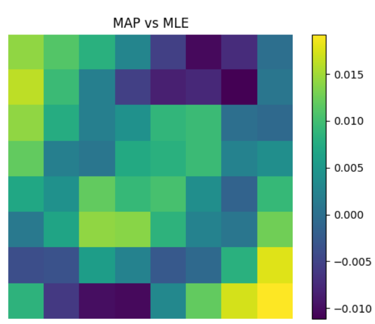

Figure 3: $\hat{\mathbf{p}}^{MAP} - \hat{\mathbf{p}}^{MLE}$

Priors can also help ensure numerical stability during calculations. The logarithm of zero is negative infinity, so having if the MLE is zero it can be problematic for log-likelihood calculations whereas MAP can ensure non-zero probabilities. Interestingly, when $\alpha = \beta = 1$, $\hat{\mathbf{p}}^{MLE} = \hat{\mathbf{p}}^{MAP}$. This is when the prior is a uniform distribution and so there is uniform bias on the location of $\mathbf{p}$ and we recover the MLE.

On the other hand, a mis-specified prior can be problematic, as the estimated parameters might be skewed by the prior and not properly represent the underlying data generating process, this can result in parameter estimates that are 'worse' than using the MLE if our data set is limited in size.

Question 2

When all D components are generated from a Bernoulli distribution with $p_d = 0.5$, we have the likelihood function for model M_1:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|\mathbf{p}^{(1)} = [0.5, 0.5, ..., 0.5]^T, M_1) = \prod_{n=1}^{N}\prod_{d=1}^{D}(0.5)^{x_d^{(n)}}(0.5)^{1-x_d^{(n)}}$$

Knowing that either $x_d^{(n)}$ or $1 - x_d^{(n)}$ will be 1 and the other zero, we can simplify $(0.5)^{x_d^{(n)}}(1 - 0.5)^{1-x_d{}^{(n)}}$ to 0.5:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|\mathbf{p}^{(1)} = [0.5, 0.5, ..., 0.5]^T, M_1) = \prod_{n=1}^{N}\prod_{d=1}^{D}(0.5)$$

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|\mathbf{p}^{(1)} = [0.5, 0.5, ..., 0.5]^T, M_1) = 0.5^{N \cdot D}$$

When all D components are generated from Bernoulli distributions with unknown, but identical, p_d, we have the likelihood function for model M_2:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|\mathbf{p}^{(2)} = [p_d, p_d, ..., p_d]^T, M_2) = \prod_{n=1}^{N}\prod_{d'=1}^{D}p_d^{x_{d'}^{(n)}}(1-p_d)^{1-x_{d'}^{(n)}}$$

When each component is Bernoulli distributed with separate, unknown p_d, we have the likelihood function for model M_3:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|\mathbf{p}^{(3)} = [p_1, p_2, ..., p_D]^T, M_3) = \prod_{n=1}^{N}\prod_{d=1}^{D}p_d^{x_d^{(n)}}(1-p_d)^{1-x_d^{(n)}}$$

For each model M_i, we can marginalise out $\mathbf{p}^{(i)}$ to get $P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_i)$:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_i) = \int_0^1 ... \int_0^1 P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|\mathbf{p}^{(i)}, M_i)P(\mathbf{p}^{(i)}|M_i)dp_1...dp_D$$

Given that the prior of any unknown probabilities is uniform, i.e. $P(\mathbf{p}^{(i)}|M_i) = 1$. We can simplify:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_i) = \int_0^1 ... \int_0^1 P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|\mathbf{p}^{(i)}, M_i)dp_1...dp_D$$

For M_1:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_1) = \int_0^1 ... \int_0^1 0.5^{N \cdot D}d\theta_1...d\theta_D$$

We can remove the integrals:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_1) = 0.5^{N \cdot D}$$

For M_2, we have that all pixels share some probability p_d so we only need to integrate over a single variable p_d:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_2) = \int_0^1 \prod_{n=1}^{N}\prod_{d'=1}^{D}p_d^{x_{d'}^{(n)}}(1-p_d)^{1-x_{d'}{}^{(n)}}dp_d$$

Changing the products to sums:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_2) = \int_0^1 p_d^{\sum_{n=1}^{N}\sum_{d'=1}^{D} x_{d'}^{(n)}}(1-p_d)^{\sum_{n=1}^{N}\sum_{d'=1}^{D} 1-x_{d'}^{(n)}}dp_d$$

Rewriting:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_2) = \int_0^1 (p_d)^K(1-p_{d=1})^{N\cdot D-K}dp_d$$

where $K = \sum_{n=1}^{N}\sum_{d'=1}^{D} x_{d'}^{(n)}$.
This integral is the beta function:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_2) = \frac{K!(N\cdot D-K)!}{(N\cdot D+1)!}$$

For M_3, we need an integral for each p_d:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_3) = \int_0^1 ... \int_0^1 \prod_{n=1}^{N}\prod_{d=1}^{D} p_d^{x_d^{(n)}}(1-p_d)^{1-x_d{}^{(n)}}dp_1...dp_D$$

We can separate the integrals to only contain the relevant p_d:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_3) = \prod_{d=1}^{D}\left(\int_0^1 \prod_{n=1}^{N} p_d^{x_d^{(n)}}(1-p_d)^{1-x_d{}^{(n)}}dp_d\right)$$

Changing the products to sums:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_3) = \prod_{d=1}^{D}\left(\int_0^1 p_d^{\sum_{n=1}^{N} x_d^{(n)}}(1-p_d)^{\sum_{n=1}^{N} 1-x_d{}^{(n)}}dp_d\right)$$

In this case, we have the product of integrals where each evaluates to a beta function:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_3) = \prod_{d=1}^{D}\frac{K_d!(N-K_d)!}{(N+1)!}$$

where $K_d = \sum_{n=1}^{N} x_d^{(n)}$.
The posterior probability of a model M_i can be expressed:

$$P(M_i|\{\mathbf{x}^{(n)}\}_{n=1}^{N}) = \frac{P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_i)P(M_i)}{P(\{\mathbf{x}^{(n)}\}_{n=1}^{N})}$$

We only have three models, so in this case the normalisation $P(\{\mathbf{x}^{(n)}\}_{n=1}^{N})$ can be expressed as a sum:

$$P(M_i|\{\mathbf{x}^{(n)}\}_{n=1}^{N}) = \frac{P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_i)P(M_i)}{\sum_{i\in\{1,2,3\}} P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_i)P(M_i)}$$

Given that $P(M_i) = \frac{1}{3}$ for all $i \in \{1,2,3\}$:

$$P(M_i|\{\mathbf{x}^{(n)}\}_{n=1}^{N}) = \frac{P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_i)}{\sum_{i\in\{1,2,3\}} P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|M_i)}$$

Calculating the posterior probabilities of each of the three models having generated the data in binarydigits.txt using Python, we can show the values in the Table 1.

i	$P(M_i \| \{\mathbf{x}^{(n)}\}_{n=1}^N)$
1	1E-1924
2	1E-1858
3	1-(1E-1924)-(1E-1858)

Table 1: Posterior Probabilities

We can see that for models specified to have the same parameter value for all pixels, like M_1, are very unlikely with the given data set. This makes sense because it is specifying models where the image is essentially a uniform shade, which is not reflective of our digit images. Moreover, M_1 specifies a specific value of 0.5 for all the parameters whereas M_2 specifies any value for all the parameters as long as it's the same. So the model M_1 is just one possible model specified in M_2 and we can see this reflected in our probabilities when $P(M_2|\{\mathbf{x}^{(n)}\}_{n=1}^N) > P(M_1|\{\mathbf{x}^{(n)}\}_{n=1}^N)$.

The Python code for calculating the posterior probabilities of the three models:

```python
import numpy as np
import pandas as pd
from scipy.special import betaln, logsumexp


def _log_p_d_given_m1(x: np.ndarray) -> float:
    """
    Calculates log likelihood of model 1
    :param x: numpy array of shape (N, D)
    :return: log likelihood
    """
    n, d = x.shape
    return n * d * np.log(0.5)


def _log_p_d_given_m2(x: np.ndarray):
    """
    Calculates log likelihood of model 2
    :param x: numpy array of shape (N, D)
    :return: log likelihood
    """
    n, d = x.shape
    k = np.sum(x).astype(int)
    return betaln(k + 1, n * d - k + 1)


def _log_p_d_given_m3(x: np.ndarray):
    """
    Calculates log likelihood of model 3
    :param x: numpy array of shape (N, D)
    :return: log likelihood
    """
    n, _ = x.shape
    k_d = np.sum(x, axis=0).astype(int)
    return logsumexp(betaln(k_d + 1, n - k_d + 1))


def _log_p_model_given_data(x) -> np.ndarray:
    """
    Calculates posterior log likelihood of models given image data
    :param x: numpy array of shape (N, D)
    :return: posterior log likelihood
    """
    log_p_d_given_m = np.array(
        [
            _log_p_d_given_m1(x),
            _log_p_d_given_m2(x),
            _log_p_d_given_m3(x),
        ]
    )
    log_p_m_given_data = log_p_d_given_m - logsumexp(log_p_d_given_m)
    return log_p_m_given_data


def c(x: np.ndarray, table_path: str) -> None:
    """
    Produces answers for question 2c
    :param x: numpy array of shape (N, D)
    :param table_path: path to store table posterior likelihoods
    :return:
    """
    log_p_m_given_data = _log_p_model_given_data(x)
    df = pd.DataFrame(
        data=np.array(
            [
                np.arange(len(log_p_m_given_data)).astype(int) + 1,
                [f"1E{int(x/np.log(10))}" for x in log_p_m_given_data[:-1]]
                + [
                    f"1-{'-'.join([f'(1E{int(x/np.log(10))})' for x in log_p_m_given_data[:-1]])}"
                ],
            ]
        ).T,
        columns=["Model", "P(M_i|D)"],
    )
    df.set_index("Model", inplace=True)
    df.to_csv(table_path)
```

src/solutions/q2.py

Question 3

(a) The likelihood for a model consisting of a mixture of K multivariate Bernoulli distributions can be expressed as the product across N data points:

$$P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|\theta) = \prod_{i=1}^{N} P(\mathbf{x}^{(n)}|\theta)$$

where $\{\mathbf{x}^{(n)}\}_{n=1}^{N}$ is our data set with $\mathbf{x}^{(n)} \in \mathbb{R}^{D\times 1}$ and $\theta = \{\pi, \mathbf{P}\}$ are our parameters, $\pi = [\pi_1, ..., \pi_K] \in \mathbb{R}^{K\times 1}$ our K mixing proportions ($0 \leq \pi_k \leq 1; \sum_k \pi_k = 1$) and $\mathbf{P} \in \mathbb{R}^{D\times K}$ the K Bernoulli parameter vectors with elements p_{kd} denoting the probability that pixel d takes value 1 given mixture component k. We also assume the images are iid and that the pixels are independent of each other within each component distribution.

For each $P(\mathbf{x}^{(n)}|\theta)$:

$$P(\mathbf{x}^{(n)}|\theta) = \sum_{k=1}^{K} \pi_k \prod_{d=1}^{D} (p_{kd})^{x_d^{(n)}} (1 - p_{kd})^{1-x_d^{(n)}}$$

The log-likelihood $\mathcal{L}(\mathbf{x}^{(n)}|\theta)$ can be expressed in vector form:

$$\mathcal{L}(\mathbf{x}^{(n)}|\theta) = \log \sum_{k=1}^{K} \pi_k \exp\Big((\mathbf{x}^{(n)})^T \log(\mathbf{P}_k) + (1 - \mathbf{x}^{(n)})^T \log(1 - \mathbf{P}_k)\Big)$$

where $\mathbf{P}_k$ is the k^{th} column of $\mathbf{P}$. This can can be further vectorised using Python scipy's *logsumexp* operation.

Moreover, the log-likelihood $\mathcal{L}(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|\theta)$ can be expressed:

$$\mathcal{L}(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|\theta) = \sum_{i=1}^{N} \Bigg(\log \sum_{k=1}^{K} \pi_k \exp\Big((\mathbf{x}^{(n)})^T \log(\mathbf{P}_k) + (1 - \mathbf{x}^{(n)})^T \log(1 - \mathbf{P}_k)\Big)\Bigg)$$

(b) We know that:

$$P(A|B) \propto P(B|A)P(A)$$

Thus,

$$P(s^{(n)} = k|\mathbf{x}^{(n)}, \pi, \mathbf{P}) \propto P(\mathbf{x}^{(n)}|s^{(n)} = k, \pi, \mathbf{P})P(s^{(n)} = k|\pi, \mathbf{P})$$

where $s^{(n)} \in \{1, ..., K\}$ a discrete hidden variable with $P(s^{(n)} = k|\pi) = \pi_k$. Note that $P(s^{(n)} = k|\pi, \mathbf{P}) = P(s^{(n)} = k|\pi)$ as $s^{(n)} = k$ isn't dependent on $\mathbf{P}$.

Let $\tilde{r}_{nk}$ be the unnormalised responsibility $P(\mathbf{x}^{(n)}|s^{(n)} = k, \pi, \mathbf{P})P(s^{(n)} = k|\pi, \mathbf{P})$. Using the mixture for component k, π_k and the likelihood function of component k:

$$\tilde{r}_{nk} = \pi_k \prod_{d=1}^{D} (p_{kd})^{x_d^{(n)}} (1 - p_{kd})^{1-x_d^{(n)}}$$

Normalising across the components:

$$r_{nk} = \frac{\tilde{r}_{nk}}{\sum_{j=1}^{K} \tilde{r}_{nj}}$$

we have calculated $P(s^{(n)} = k|\mathbf{x}^{(n)}, \pi, \mathbf{P})$ for the E step of an EM algorithm.

Moreover,

$$\log \tilde{r}_{nk} = \log \pi_k + \sum_{d=1}^{D} \left(x_d^{(n)} \log(p_{kd}) + (1 - x_d^{(n)}) \log(1 - \exp(\log(p_{kd}))) \right)$$

and

$$\log r_{nk} = \log \tilde{r}_{nk} - \log \sum_{j=1}^{K} \exp(\log \tilde{r}_{nj})$$

which can be vectorised as $\log \mathbf{r}$ calculated with $\log \pi$ and $\log \mathbf{P}$ using Python scipy's *logsumexp* operation.

(c) We know that the expectation log joint can be expressed:

$$\left\langle \sum_n \log P(\mathbf{x}^{(n)}, s^{(n)}|\pi, \mathbf{P}) \right\rangle_{q(\{s^{(n)}\})} = \sum_{n=1}^{N} q(s^{(n)}) \log P(\mathbf{x}^{(n)}, s^{(n)}|\pi, \mathbf{P})$$

Let this quantity be E. For each term of the summation in E:

$$q(s^{(n)}) = \mathbf{r}_n^T$$

and

$$\log P(\mathbf{x}^{(n)}, s^{(n)}|\pi, \mathbf{P}) = \log[P(\mathbf{x}^{(n)}|s^{(n)}, \pi, \mathbf{P})P(s^{(n)}|\pi, \mathbf{P})]$$

which is the vectorised version of $\log \tilde{r}_{nk}$ from part (b) so:

$$\log P(\mathbf{x}^{(n)}, s^{(n)}|\pi, \mathbf{P}) = \log(\pi) + \log(\mathbf{P})^T \mathbf{x}^{(n)} + \log(1 - \mathbf{P})^T (1 - \mathbf{x}^{(n)})$$

Combining:

$$E = \sum_n \mathbf{r}_n^T [\log(\pi) + \log(\mathbf{P})^T \mathbf{x}^{(n)} + \log(1 - \mathbf{P})^T (1 - \mathbf{x}^{(n)})]$$

To maximise with respect to π and $\mathbf{P}$ for the M step, we want to take the derivative, set to zero, and solve for $\hat{\pi}$ and $\hat{\mathbf{P}}$.

For the k^{th} element of π:

$$\frac{\partial E}{\partial \pi_k} = \sum_n r_{nk} \frac{1}{\pi_k}$$

We can calculate the maximiser with:

$$\frac{\partial E}{\partial \pi_k} + \lambda = 0$$

where λ is a Lagrange multiplier ensuring that the mixing proportions sum to unity.

Thus,

$$\hat{\pi}_k = \frac{\sum_n r_{nk}}{N}$$

For the dk^{th} element of $\mathbf{P}$:

$$\frac{\partial E}{\partial \mathbf{P}_{dk}} = \sum_n r_{nk} \frac{\partial}{\partial \mathbf{P}_{dk}} [x_d^{(n)} \log \mathbf{P}_{dk} + (1 - x_d^{(n)}) \log(1 - \mathbf{P}_{dk})]$$

Simplifying:

$$\frac{\partial E}{\partial \mathbf{P}_{dk}} = \sum_n r_{nk} \left(\frac{x_d^{(n)}}{\mathbf{P}_{dk}} - \frac{1 - x_d^{(n)}}{1 - \mathbf{P}_{dk}}\right)$$

Setting the derivative to zero:

$$\frac{\sum_n x_d^{(n)} r_{nk}}{\hat{\mathbf{P}}_{dk}} - \frac{\sum_n r_{nk} - \sum_n x_d^{(n)} r_{nk}}{1 - \hat{\mathbf{P}}_{dk}} = 0$$

Solving for $\hat{\mathbf{P}}_{dk}$:

$$\hat{\mathbf{P}}_{dk} \sum_n r_{nk} - \hat{\mathbf{P}}_{dk} \sum_n x_d^{(n)} r_{nk} = \sum_n x_d^{(n)} r_{nk} - \hat{\mathbf{P}}_{dk} \sum_n x_d^{(n)} r_{nk}$$

Thus,

$$\hat{\mathbf{P}}_{dk} = \frac{\sum_n x_d^{(n)} r_{nk}}{\sum_n r_{nk}}$$

We have the maximizing parameters for the expected log-joint

$$\arg\max_{\pi, \mathbf{P}} \left\langle \sum_n \log P(\mathbf{x}^{(n)}, s^{(n)} | \pi, \mathbf{P}) \right\rangle_{q(\{s^{(n)}\})}$$

thus obtaining an iterative update for the parameters π and $\mathbf{P}$ in the M-step of EM.

For numerical stability, we can compute the maximisation step for the MAP of $\mathbf{P}$, by solving for $\hat{\mathbf{P}}_{dk}^{MAP}$ with:

$$\frac{\partial E'}{\partial \mathbf{P}_{dk}} = 0$$

where

$$E' = \sum_{n=1}^{N} q(s^{(n)}) \log P(\mathbf{P} | \pi, \mathbf{x}^{(n)}, s^{(n)})$$

and from Bayes':

$$\log P(\mathbf{P} | \pi, \mathbf{x}^{(n)}, s^{(n)}) = \log P(\mathbf{x}^{(n)}, s^{(n)} | \pi, \mathbf{P}) + \log P(\mathbf{P}) - \log P(\mathbf{x}^{(n)}, s^{(n)} | \pi)$$

Assuming an independent Beta prior on each pixel of each component:

$$\log P(\mathbf{P}) = \sum_{k=1}^{K} \sum_{d=1}^{D} - \log(B(\alpha, \beta)) + (\alpha - 1) \log \mathbf{P}_{dk} + (\beta - 1) \log(1 - \mathbf{P}_{dk})$$

and

$$\frac{\partial \log P(\mathbf{P})}{\partial \mathbf{P}_{dk}} = \frac{(\alpha - 1)}{\mathbf{P}_{dk}} - \frac{(\beta - 1)}{1 - \mathbf{P}_{dk}}$$

Thus, the derivative can be expressed as:

$$\frac{\partial E'}{\partial \mathbf{P}_{dk}} = \sum_n \left(r_{nk} \left(\frac{\partial \log P(\mathbf{x}^{(n)}, s^{(n)} | \pi, \mathbf{P})}{\partial \mathbf{P}_{dk}} + \frac{\partial \log P(\mathbf{P})}{\partial \mathbf{P}_{dk}} \right) \right)$$

Substituting the appropriate expressions:

$$\frac{\partial E'}{\partial \mathbf{P}_{dk}} = \sum_n \left(r_{nk} \Big(\frac{x_d^{(n)}}{\mathbf{P}_{dk}} - \frac{1 - x_d^{(n)}}{1 - \mathbf{P}_{dk}} + \frac{(\alpha - 1)}{\mathbf{P}_{dk}} - \frac{(\beta - 1)}{1 - \mathbf{P}_{dk}} \Big) \right)$$

Simplifying:

$$\frac{\partial E'}{\partial \mathbf{P}_{dk}} = \frac{\sum_n r_{nk}(\alpha - 1 + x_d^{(n)})}{\mathbf{P}_{dk}} - \frac{\sum_n r_{nk}(\beta - x_d^{(n)})}{1 - \mathbf{P}_{dk}}$$

Setting $\frac{\partial E'}{\partial \mathbf{P}_{dk}} = 0$ we can calculate $\hat{\mathbf{P}}_{dk}^{MAP}$:

$$\sum_n r_{nk}(\alpha - 1 + x_d^{(n)}) - \hat{\mathbf{P}}_{dk} \sum_n r_{nk}(\alpha - 1 + x_d^{(n)}) = \hat{\mathbf{P}}_{dk} \sum_n r_{nk}(\beta - x_d^{(n)})$$

$$\hat{\mathbf{P}}_{dk}^{MAP} = \frac{\sum_n r_{nk}(x_d^{(n)} + \alpha - 1)}{(\alpha + \beta - 1)(\sum_n r_{nk})}$$

As a sense check, we can see when setting $\alpha = 1$ and $\beta = 1$ we recover $\hat{\mathbf{P}}_{dk}^{MLE}$ as we would expect. For the following parts, a very weak Beta(1+1e-5, 1+1e-5) prior was used.

(d) Plotting the unnormalised posterior likelihood as a function of the iteration number for different k values:

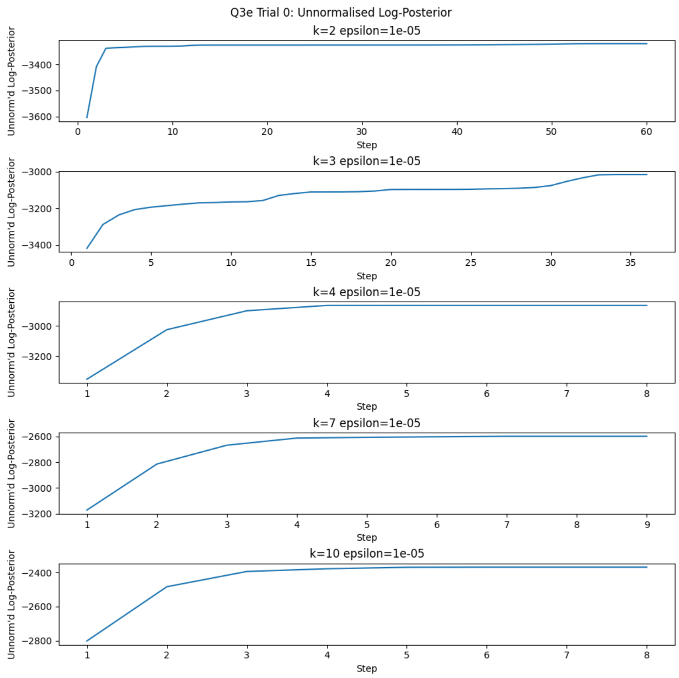

Figure 4: Unnormalised Log Posterior vs Iteration Number

where *epsilon* is the stopping condition for when the unnormalised log posterior converges sufficiently. Note that the normalisation constant for the log posterior $\log P(\mathbf{x}^{(n)}, s^{(n)}|\pi)$ is intractable and so only the unnormalised portion $\log P(\mathbf{x}^{(n)}, s^{(n)}|\pi, \mathbf{P}) + \log P(\mathbf{P})$ was computed and reported.

Displaying the parameters found for $K \in \{2, 3, 4, 7, 10\}$:

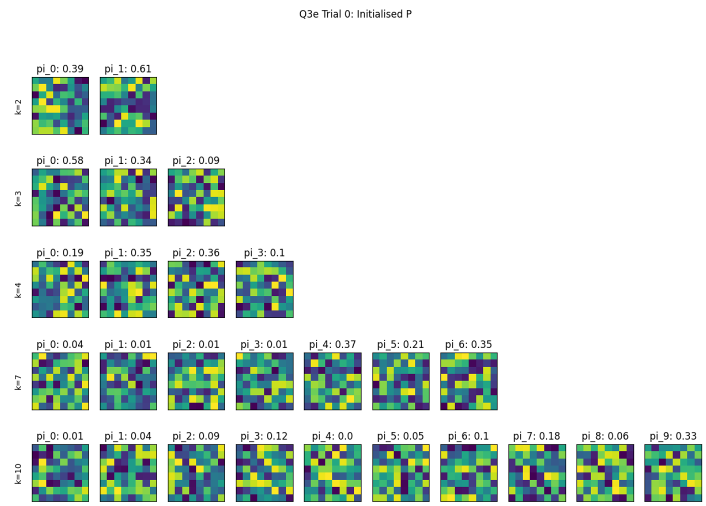

Figure 5: Randomly initialised parameters

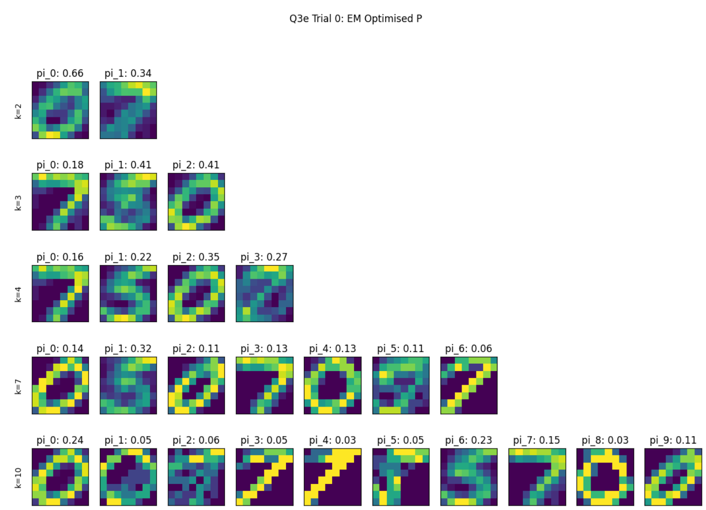

Figure 6: EM optimised parameters

The Python code for the EM algorithm:

```python
from dataclasses import dataclass
from typing import List, Tuple

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.special import betaln, logsumexp
from sklearn.manifold import TSNE

from src.constants import DEFAULT_SEED


@dataclass
class Theta:
    """
    Data class containing the model parameters
    log_pi: the logarithm of the mixing proportions (1, k)
    log_p_matrix: the logarithm of the probability where the (i,j)th element is the probability that
                  pixel j takes value 1 under mixture component i (d, k)
    """

    log_pi: np.ndarray
    log_p_matrix: np.ndarray

    @property
    def pi(self) -> np.ndarray:
        """
        Calculates the mixing proportions
        :return: vector of mixing proportions (1, k)
        """
        return np.exp(self.log_pi)

    @property
    def p_matrix(self) -> np.ndarray:
        """
        Calculates the Bernoulli parameters
        :return: matrix Bernoulli parameters (d, k)
        """
        d, k = self.log_p_matrix.shape
        image_dimension = int(np.sqrt(d))
        return np.exp(self.log_p_matrix).reshape(image_dimension, image_dimension, -1)

    @property
    def log_one_minus_p_matrix(self) -> np.ndarray:
        """
        Compute log(1-P) where P=exp(log_p_matrix)
        :return: an array of the same shape as log_p_matrix (d, k)
        """
        log_of_one = np.zeros(self.log_p_matrix.shape)
        stacked_sum = np.stack((log_of_one, self.log_p_matrix))
        weights = np.ones(stacked_sum.shape)
        weights[1] = -1  # scale p matrix by -1 for subtraction
        return np.array(logsumexp(stacked_sum, b=weights, axis=0))

    def log_pi_repeated(self, n: int) -> np.ndarray:
        """
        Repeats the log_pi vector n times along axis 0
        :param n: number of repetitions
        :return: an array of shape (n, k)
        """
        return np.repeat(self.log_pi, n, axis=0)


def _init_params(k: int, d: int) -> Theta:
    """
    Random initialisation of theta parameters (log_pi and log_p_matrix)
    :param k: Number of components
    :param d: Image dimension (number of pixels in a single image)
    :return: theta: the parameters of the model
    """
    return Theta(
        log_pi=np.log(np.random.dirichlet(np.ones(k), size=1)),
        log_p_matrix=np.log(np.random.uniform(low=0, high=1, size=(d, k))),
    )


def _compute_log_component_p_x_i_given_theta(x: np.ndarray, theta: Theta) -> np.ndarray:
    """
    Compute the unweighted probability of each mixing component for each image
    :param x: the image data (n, d)
    :param theta: the parameters of the model
    :return: an array of the unweighted probabilities (n, k)
    """
    return x @ theta.log_p_matrix + (1 - x) @ theta.log_one_minus_p_matrix


def _compute_log_p_x_i_given_theta(x: np.ndarray, theta: Theta) -> np.ndarray:
    """
    Computes the log likelihood of each image in the dataset x
    :param x: the image data (n, d)
    :param theta: the parameters of the model
    :return: log_p_x_i_given_theta: a log likelihood array containing the log likelihood of each image (n
    ,1)
    """
```

```python
    n, _ = x.shape
    log_component_probabilities = _compute_log_component_p_x_i_given_theta(
        x, theta
    )  # (n, k)
    return np.array(
        logsumexp(
            log_component_probabilities
            + theta.log_pi_repeated(n),  # scale each component by component probability
            axis=1,
        )
    )


def _compute_log_likelihood(x: np.ndarray, theta: Theta) -> float:
    """
    Computes the log likelihood of all images in the dataset x
    :param x: the image data (n, d)
    :param theta: the parameters of the model
    :return: log_p_x_given_theta: the log likelihood array across all images
    """
    return np.sum(_compute_log_p_x_i_given_theta(x, theta)).item()


def _compute_log_prior(
    theta: Theta, alpha_parameter: float, beta_parameter: float
) -> float:
    """
    Compute the prior log probability of the P matrix under a Beta prior
    :param theta: the parameters of the model
    :param alpha_parameter: alpha parameter of the beta prior
    :param beta_parameter: beta parameter of the beta prior
    :return: log_p_of_p_matrix
    """
    return np.sum(
        -betaln(alpha_parameter, beta_parameter)
        + (alpha_parameter - 1) * theta.log_p_matrix
        + (beta_parameter - 1) * theta.log_one_minus_p_matrix
    ).item()


def _compute_unnormalised_log_posterior_likelihood(
    x: np.ndarray, theta: Theta, alpha_parameter: float, beta_parameter: float
) -> float:
    """
    Compute the unnormalised posterior log probability of the P matrix
    :param x: the image data (n, d)
    :param theta: the parameters of the model
    :param alpha_parameter: alpha parameter of the beta prior
    :param beta_parameter: beta parameter of the beta prior
    :return: log_p_of_p_matrix
    """
    log_likelihood = _compute_log_likelihood(x, theta)
    log_prior = _compute_log_prior(theta, alpha_parameter, beta_parameter)
    return log_likelihood + log_prior


def _compute_log_e_step(x: np.ndarray, theta: Theta) -> np.ndarray:
    """
    Compute the e step of expectation maximisation
    :param x: the image data (n, d)
    :param theta: the parameters of the model
    :return: an array of the log responsibilities of k mixture components for each image (n, k)
    """
    log_r_unnormalised = _compute_log_component_p_x_i_given_theta(x, theta)
    log_r_normaliser = logsumexp(log_r_unnormalised, axis=1)
    log_responsibility = log_r_unnormalised - log_r_normaliser[:, np.newaxis]
    return log_responsibility


def _compute_log_pi_hat(log_responsibility: np.ndarray) -> np.ndarray:
    """
    Compute the log of the maximised mixing proportions
    :param log_responsibility: an array of the log responsibilities of k mixture components for each image
     (n, k)
    :return: an array of the maximised log mixing proportions (1, k)
    """
    n, _ = log_responsibility.shape
    return (logsumexp(log_responsibility, axis=0) - np.log(n)).reshape(1, -1)


def _compute_log_p_matrix_hat(
    x: np.ndarray,
    log_responsibility: np.ndarray,
    alpha_parameter: float,
    beta_parameter: float,
) -> np.ndarray:
    """
    Compute the log of the maximised pixel probabilities
    :param x: the image data (n, d)
    :param log_responsibility: an array of the log responsibilities of k mixture components for each image
     (n, k)
    :param alpha_parameter: alpha parameter of the beta prior
    :param beta_parameter: beta parameter of the beta prior
    :return: an array of the maximised pixel probabilities for each component (d, k)
    """
    n, d = x.shape
```

```python
    _, k = log_responsibility.shape

    x_repeated = np.repeat(x[:, :, np.newaxis], k, axis=2)  # (n, d, k)
    log_responsibility_repeated = np.repeat(
        log_responsibility[:, np.newaxis, :], d, axis=1
    )  # (n, d, k)

    log_p_matrix_unnormalised_posterior = logsumexp(
        log_responsibility_repeated, b=(x_repeated + alpha_parameter - 1), axis=0
    )  # (d, k)

    log_p_matrix_normaliser_posterior = logsumexp(
        log_responsibility_repeated, b=(alpha_parameter + beta_parameter - 1), axis=0
    )  # (d, k)

    log_p_matrix_normalised_posterior = (
        log_p_matrix_unnormalised_posterior - log_p_matrix_normaliser_posterior
    )  # (d, k)
    return log_p_matrix_normalised_posterior


def _compute_log_m_step(
    x: np.ndarray,
    log_responsibility: np.ndarray,
    alpha_parameter: float,
    beta_parameter: float,
) -> Theta:
    """
    Compute the m step of expectation maximisation
    :param x: the image data (n, d)
    :param log_responsibility: an array of the log responsibilities of k mixture components for each image (n, k)
    :param alpha_parameter: alpha parameter of the beta prior
    :param beta_parameter: beta parameter of the beta prior
    :return: thetas optimised after maximisation step
    """
    return Theta(
        log_pi=_compute_log_pi_hat(log_responsibility),
        log_p_matrix=_compute_log_p_matrix_hat(
            x, log_responsibility, alpha_parameter, beta_parameter
        ),
    )


def _run_expectation_maximisation(
    x: np.ndarray,
    theta: Theta,
    alpha_parameter: float,
    beta_parameter: float,
    max_number_of_steps: int,
    epsilon: float,
) -> Tuple[Theta, np.ndarray, List[float]]:
    """
    Run the expectation maximisation algorithm
    :param x: the image data (n, d)
    :param theta: initial theta parameters
    :param alpha_parameter: alpha parameter of the beta prior
    :param beta_parameter: beta parameter of the beta prior
    :param max_number_of_steps: the maximum number of steps to run the algorithm
    :param epsilon: the minimum required change in log posterior, otherwise the algorithm stops early
    :return: a tuple containing the optimised thetas, the log responsibilities,
             and the log log_posteriors at each step of the algorithm
    """
    log_responsibility = None
    log_posteriors = []
    for _ in range(max_number_of_steps):
        log_responsibility = _compute_log_e_step(x, theta)
        theta = _compute_log_m_step(
            x, log_responsibility, alpha_parameter, beta_parameter
        )

        log_posteriors.append(
            _compute_unnormalised_log_posterior_likelihood(
                x, theta, alpha_parameter, beta_parameter
            )
        )

        #  check for early stopping
        if len(log_posteriors) > 1:
            if (log_posteriors[-1] - log_posteriors[-2]) < epsilon:
                break
    return theta, log_responsibility, log_posteriors


def _visualise_p_matrix(
    thetas: List[Theta], ks: List[int], figure_title: str, figure_path: str
) -> None:
    """
    Visualises the P matrix for different thetas and ks
    :param thetas: list of Theta instances
    :param ks: list of k values used for each Theta
    :param figure_title: name of figure
    :param figure_path: path to store figure
    :return:
    """
    n = len(ks)
```

```python
    m = np.max(ks)
    fig = plt.figure()
    fig.set_figwidth(15)
    fig.set_figheight(10)
    for i, k in enumerate(ks):
        for j in range(k):
            ax = plt.subplot(n, m, m * i + j + 1)
            ax.imshow(
                thetas[i].p_matrix[:, :, j],
                interpolation="None",
            )
            ax.tick_params(
                axis="x",
                which="both",
                bottom=False,
                top=False,
            )
            ax.tick_params(
                axis="y",
                which="both",
                left=False,
                right=False,
            )
            ax.xaxis.set_ticklabels([])
            ax.yaxis.set_ticklabels([])
            ax.set_title(f"pi_{j}: {np.round(thetas[i].pi[0, j], 2)}")
            if j == 0:
                ax.set_ylabel(f"{k=}")
    fig.suptitle(figure_title)
    plt.savefig(figure_path)


def _visualise_responsibility_clusters(
    log_responsibilities: List[np.ndarray],
    ks: List[int],
    figure_title: str,
    figure_path: str,
) -> None:
    """
    Visualise responsibility vectors of images using TSNE for different k values
    :param log_responsibilities: list of log responsibilities for different ks
    :param ks: list of k values used for each Theta
    :param figure_title: name of figure
    :param figure_path: path to store figure
    :return:
    """
    n = len(ks)
    fig = plt.figure()
    fig.set_figwidth(5 * n)
    fig.set_figheight(5)
    for i, k in enumerate(ks):
        if k > 2:
            # use TSNE when we have more than 2 dimensions
            embedding = TSNE(
                n_components=2,
                learning_rate="auto",
                init="random",
                perplexity=10,
                random_state=DEFAULT_SEED,
            ).fit_transform(log_responsibilities[i])
        else:
            # otherwise we can visualise responsibility vectors without dimensionality reduction
            embedding = np.exp(log_responsibilities[i])
        ax = plt.subplot(1, n, i + 1)
        ax.scatter(embedding[:, 0], embedding[:, 1])
        ax.set_title(f"{k=}")
    fig.suptitle(figure_title)
    plt.savefig(figure_path, bbox_inches="tight")


def _plot_log_posteriors(
    log_posteriors: List[List[float]],
    ks: List[int],
    epsilon: float,
    figure_title: str,
    figure_path: str,
) -> None:
    """
    Plot log posteriors as a function of EM iteration for different ks
    :param log_posteriors: list of vectors, each representing the log posterior during EM for a specific k
    :param ks: list of k values used for each Theta
    :param epsilon: value used for early stopping of EM
    :param figure_title: name of figure
    :param figure_path: path to store figure
    :return:
    """
    fig, ax = plt.subplots(len(ks), 1, constrained_layout=True)
    fig.set_figwidth(10)
    fig.set_figheight(10)
    for i, k in enumerate(ks):
        ax[i].plot(np.arange(1, len(log_posteriors[i]) + 1), log_posteriors[i])
        ax[i].set_xlabel("Step")
        ax[i].set_ylabel(f"Unnorm'd Log-Posterior")
        ax[i].set_title(f"{k=} {epsilon=}")
    plt.suptitle(figure_title)

```

```python
    plt.savefig(figure_path)


def _compute_compression_rate(
    ks: List[int], log_posteriors: List[List[float]], i: int, n: int, d: int
) -> pd.DataFrame:
    """
    Compute the compress rate, not taking into account the cost of storing model parameters
    :param ks: k values to use for each trial
    :param log_posteriors: list of vectors, each representing the log posterior during EM for a specific k
    :param i: trial number
    :param n: number of data points
    :param d: number of dimensions per data point
    :return: dataframe containing the compression rate for this trial
    """
    df = pd.DataFrame(
        data=[
            [
                np.round(1 - (-log_posterior[-1] / (np.log(2) * n * d)), 2)
                for log_posterior in log_posteriors
            ]
        ],
        columns=ks,
    ).T
    df = df.reset_index()
    df.columns = ["k value", f"Trial {i}"]
    return df.set_index("k value")


def _compute_total_compression_ratio(
    ks: List[int], log_posteriors: List[List[float]], i: int, n: int, d: int
) -> pd.DataFrame:
    """
    Compute the total compress ratio, taking into account the cost of storing model parameters (assuming
     float64)
    :param ks: k values to use for each trial
    :param log_posteriors: list of vectors, each representing the log posterior during EM for a specific k
    :param i: trial number
    :param n: number of data points
    :param d: number of dimensions per data point
    :return: dataframe containing the total compression ratios for this trial
    """
    df = pd.DataFrame(
        data=[
            [
                np.round(
                    (-log_posterior[-1] + (64 * ks[j] * (d + 1))) / (np.log(2) * n * d),
                    2,
                )
                for j, log_posterior in enumerate(log_posteriors)
            ]
        ],
        columns=ks,
    ).T
    df = df.reset_index()
    df.columns = ["k value", f"Trial {i}"]
    return df.set_index("k value")


def e(
    x: np.ndarray,
    alpha_parameter: float,
    beta_parameter: float,
    number_of_trials: int,
    ks: List[int],
    epsilon: float,
    max_number_of_steps: int,
    figure_path: str,
    figure_title: str,
    compression_csv_path: str,
) -> None:
    """
    Produces answers for question 3e
    :param x: numpy array of shape (N, D)
    :param alpha_parameter: alpha parameter of the beta prior
    :param beta_parameter: beta parameter of the beta prior
    :param number_of_trials: number of trails to run EM
    :param ks: k values to use for each trial
    :param epsilon: value used for early stopping of EM
    :param max_number_of_steps: maximum number of steps during EM
    :param figure_title: base name of figures
    :param figure_path: base paths to store figure
    :param compression_csv_path: path to store bits data
    :return:
    """
    n, d = x.shape
    np.random.seed(DEFAULT_SEED)
    df_compression_list: List[pd.DataFrame] = []
    df_total_compression_list: List[pd.DataFrame] = []
    for i in range(number_of_trials):
        init_thetas: List[Theta] = []
        em_thetas: List[Theta] = []
        log_posteriors: List[List[float]] = []
        log_responsibilities: List[np.ndarray] = []
        for j, k in enumerate(ks):
            init_theta = _init_params(k, d)
```

```
            em_theta, log_responsibility, log_posterior = _run_expectation_maximisation(
                x,
                theta=init_theta,
                alpha_parameter=alpha_parameter,
                beta_parameter=beta_parameter,
                epsilon=epsilon,
                max_number_of_steps=max_number_of_steps,
            )
            init_thetas.append(init_theta)
            em_thetas.append(em_theta)
            log_responsibilities.append(log_responsibility)
            log_posteriors.append(log_posterior)

        _visualise_p_matrix(
            init_thetas,
            ks,
            figure_title=f"{figure_title} Trial {i}: Initialised P",
            figure_path=f"{figure_path}-{i}-initialised-p.png",
        )
        _visualise_p_matrix(
            em_thetas,
            ks,
            figure_title=f"{figure_title} Trial {i}: EM Optimised P",
            figure_path=f"{figure_path}-{i}-optimised-p.png",
        )
        _visualise_responsibility_clusters(
            log_responsibilities,
            ks,
            figure_title=f"{figure_title} Trial {i}: TSNE Responsibility Visualisation",
            figure_path=f"{figure_path}-{i}-tsne.png",
        )
        _plot_log_posteriors(
            log_posteriors,
            ks,
            epsilon,
            figure_title=f"{figure_title} Trial {i}: Unnormalised Log-Posterior",
            figure_path=f"{figure_path}-{i}-log-pos.png",
        )
        df_compression_list.append(
            _compute_compression_rate(ks, log_posteriors, i, n, d)
        )
        df_total_compression_list.append(
            _compute_total_compression_ratio(ks, log_posteriors, i, n, d)
        )
    pd.concat(df_compression_list, axis=1).to_csv(f"{compression_csv_path}.csv")
    pd.concat(df_total_compression_list, axis=1).to_csv(
        f"{compression_csv_path}-total.csv"
    )
```

src/solutions/q3.py

(e) Running the algorithm a few times starting from randomly chosen initial conditions and visualising the parameters:

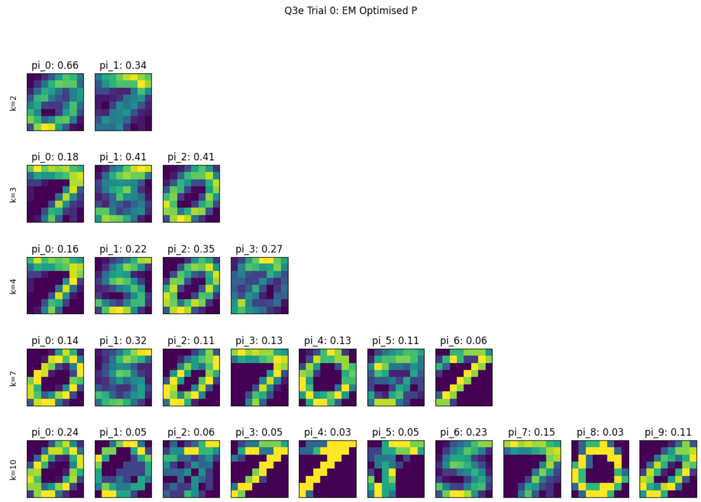

Figure 7: EM optimised parameters: Trial 0

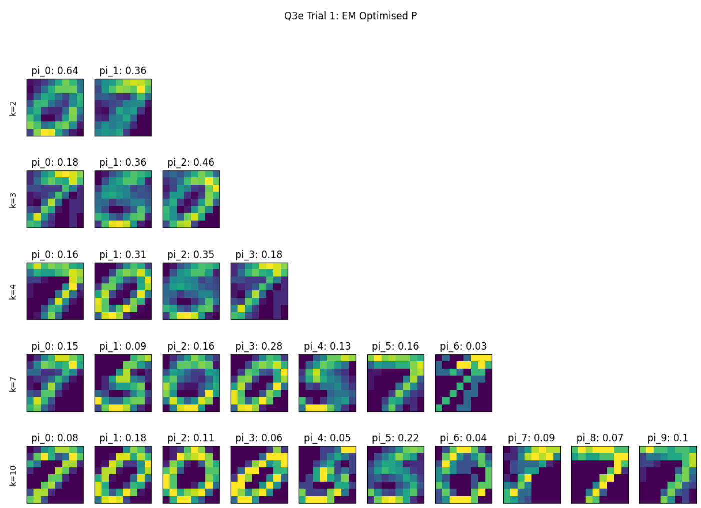

Figure 8: EM optimised parameters: Trial 1

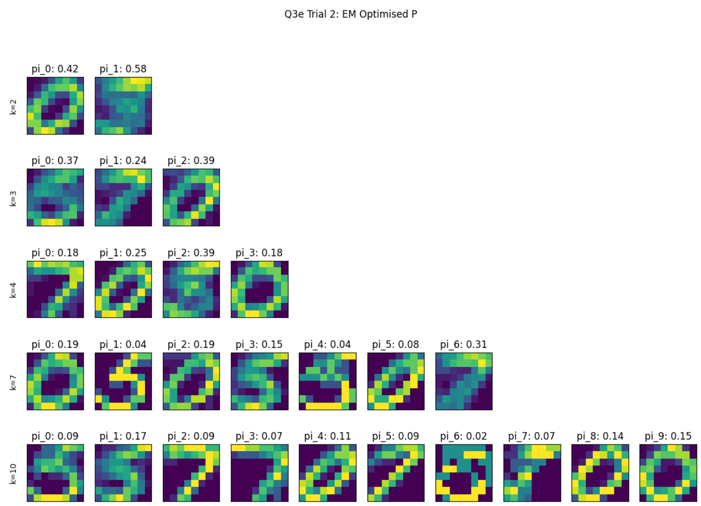

Figure 9: EM optimised parameters: Trial 2

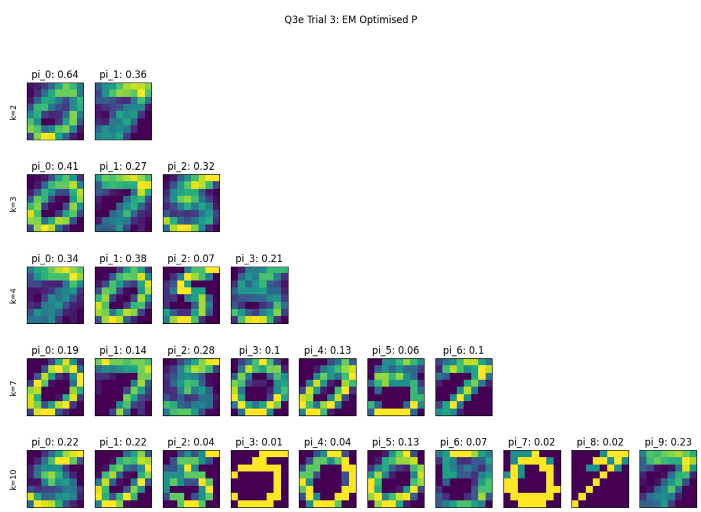

Figure 10: EM optimised parameters: Trial 3

For smaller k, we can visually see that we obtain very similar solutions (a seven and a zero for $k = 2$, although for trial 2, it looks more like zero and five mixed with seven). For $k = 3$, we get one each of zero, seven, and five, but in different permutations for different trials. However for higher k, we see that this may not always be the case. For Trial 1 of $k = 10$, we have two 5's whereas in Trial 3 we have four 5's. Interestingly, different clusters of the same digits can be different, representing different variants of the written digit (i.e. a slanted zero, a slightly slanted zero, and a symmetric zero).

Moreover, looking at the responsibilities of each mixture component, we can see that when k is relatively small they are relatively evenly distributed. However for $k = 7$ and especially $k = 10$, we can see some components have very small probability (i.e. π_3 of trial 0 and $k = 10$). It will be unlikely for those components to represent very distinct clusters (i.e. the parameters for π_3 and π_4 are very similar in trial 0 and $k = 10$). Moreover, for these small probabilities, we see that the parameters are almost like binary images. This shows that there are not many images represented in these components. This can be verified when we perform a TSNE visualisation of the responsibility vector for each of the images (Note that for $k = 2$, just the responsibility vector is plotted because it is two dimensional). We can see that for large k, qualitatively the number of clusters no longer matches the k value, indicating that some mixtures are redundant. For example for $k = 7$ and $k = 10$ we can only qualitatively see three or four clusters with TSNE.

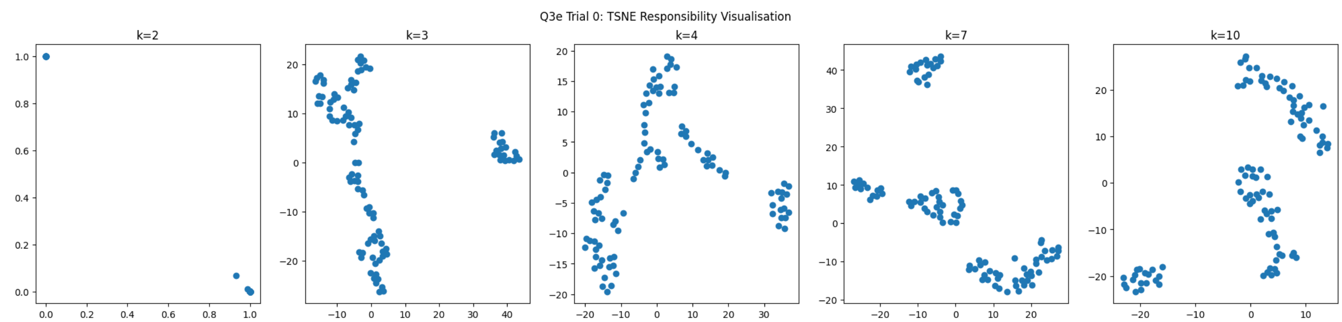

Figure 11: TSNE Visualisation of Image responsibilities: Trial 0

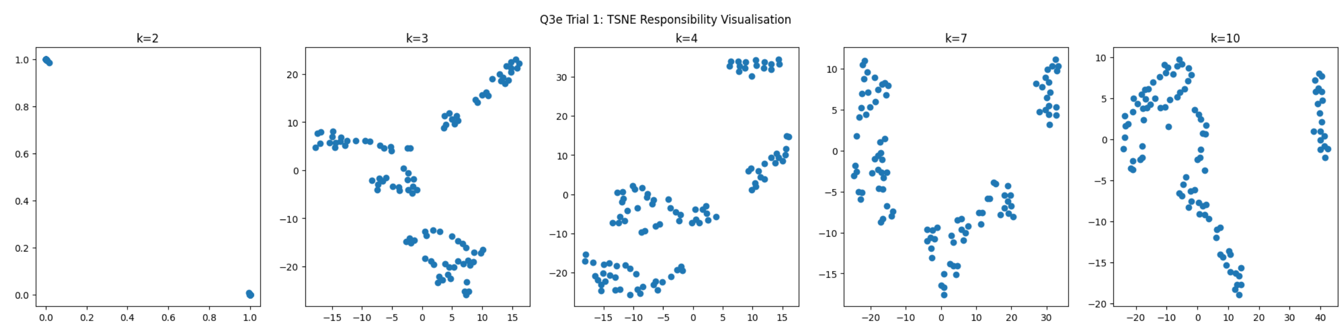

Figure 12: TSNE Visualisation of Image responsibilities: Trial 1

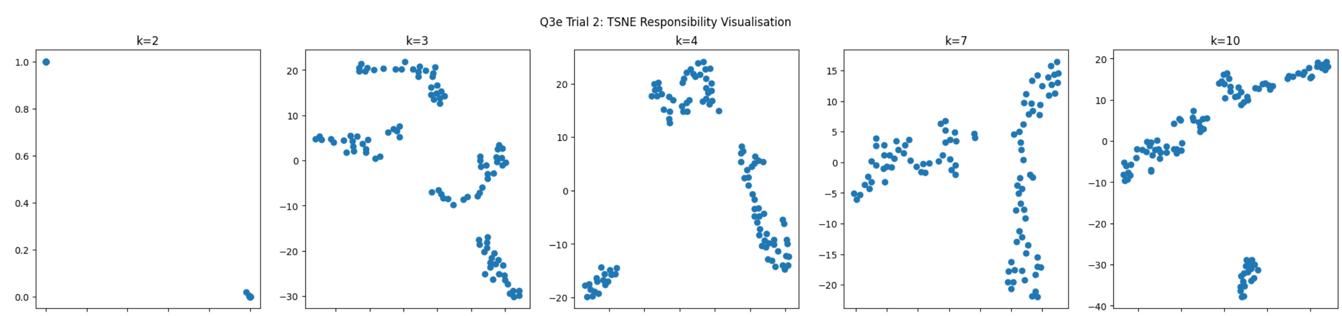

Figure 13: TSNE Visualisation of Image responsibilities: Trial 2

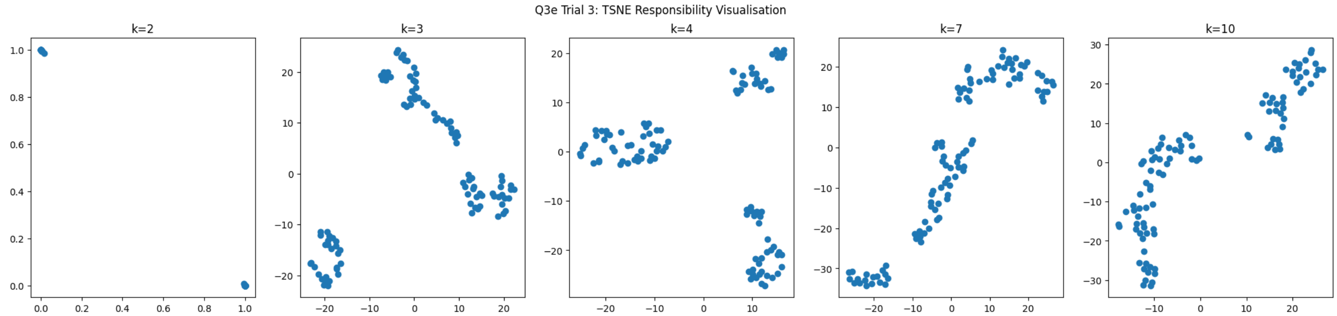

Figure 14: TSNE Visualisation of Image responsibilities: Trial 3

Improvements to the model could include searching for an optimal k by maximising the log posterior with regularisation on the magnitude of k to balance maximising log posterior with minimising model complexity. Additionally, adding a prior on the responsibility components can be helpful to ensure a more even distribution across mixture components unlike the components visualised here. This could help promote more meaningful clusters as k increases. Moreover, more experimentation for choosing better priors can be helpful to find better separation between mixtures. Increasing the size of our data set (i.e. more images) and resolution of our images (i.e. more pixels) can help the model better understand the distinguishing nuances of different mixtures and provide better clustering, although the number of images and the resolution should scale together to ensure that the model doesn't learn the noise in the higher resolution images. This is assuming we are able to scale our computing resources. Finally, given that we know that there are ten digits and our current data set only includes a subset of these digits, we can also expand our data set to include all ten digits. Hopefully, for $k = 10$, we will then be able to achieve a unique digit for each mixing component, rather than variations of repeated digits as we see now.

(f) The log-likelihood in bits can be expressed as:

$$\log_2(P(\{\mathbf{x}^{(n)}\}_{n=1}^N|\theta))$$

The length of the naive encoding of these binary data is $N \cdot D$, the number of pixels in $\{\mathbf{x}^{(n)}\}_{n=1}^N$. This is because the images are binary so each pixel can be represented with a single bit. We can compute compression rate with respect to the ratio of log-likelihood bits to the length of the naive encoding for each k:

$$rate = 1 - \frac{\log_2(P(\{\mathbf{x}^{(n)}\}_{n=1}^N|\theta))}{N \cdot D}$$

Presenting the compression rates for different trials and k values:

k value	Trial 0	Trial 1	Trial 2	Trial 3
2	0.25	0.25	0.26	0.25
3	0.32	0.28	0.29	0.31
4	0.35	0.36	0.36	0.32
7	0.41	0.42	0.38	0.43
10	0.47	0.47	0.49	0.43

Table 2: Compression Rates

As k increases, we can see that our compression rate gets better. This is intuitive because with higher k we are specifying a more complex and expressive model (i.e. with more parameters) and thus we are able to capture more of the structure of the data in the model. Thus, the bit rate, or information provided by a sample decreases with respect to the complexity of the model and thus our compression rate increases. From the source coding theorem, lossless compression algorithms are lower bounded by the entropy of the underlying data generating distribution $P(\mathbf{x})$. This is $-\sum_{x\in\mathcal{X}} P(\mathbf{x}) \log P(\mathbf{x})$. On the other hand, with EM we are maximising $\langle \log P(Z, X|\theta)\rangle_{q(X)} + H[q]$ or minimising the negative of this. When our proposal distribution P matches q, we recover $H[P]$ and we get the optimal model, the data generating model for encoding our data. This makes sense because we would be able to compress our data with the best possible distribution to represent the data. This matches the lower bound of the source coding theorem. However, because it is very unlikely that our proposal q will actually match P, our compression rate will always be worse than the optimal compression rate of $H[P]$. On the other hand, a compression algorithm would compressions on a per image basis, independent of the other images. And thus, it is able to attain a better compression rate for that image and is much closer to the source coding theorem lower bound. Depending on the data (i.e. $H[P]$ of the data), the compression rate of gzip can range from 60% to 88% (https://web.dev/optimizing-content-efficiency-optimize-encoding-and-transfer/text-compression-with-gzip), much higher than that of our models, as we expected.

(g) The total cost of encoding with model parameters and data is:

$$\log_2(P(\{\mathbf{x}^{(n)}\}_{n=1}^{N}|\theta)) + M \cdot K \cdot D + M \cdot K$$

Where M is the cost of storing a single float value, in our case we used $float64$ so 64 bits. The first term is the log-likelihood as expressed in part (f), the second term is the cost of storing $\mathbf{P}$, and the last term is the cost of storing π. The latter two terms scale with the value of k. This means that as k increases, our compression rate deteriorates. Looking at the total compression ratio $\frac{log_2(P(\{\mathbf{X}^{(n)}\}_{n=1}^{N}|\theta))+M \cdot K \cdot D+M \cdot K}{N \cdot D}$ in a table:

k value	Trial 0	Trial 1	Trial 2	Trial 3
2	2.62	2.63	2.62	2.63
3	3.49	3.54	3.52	3.5
4	4.4	4.39	4.39	4.43
7	7.15	7.14	7.18	7.14
10	9.91	9.91	9.89	9.95

Table 3: Total Compression Ratios

We can see that the ratio is greater than one, meaning that this is actually worse than the naive encoding. This is due to the high cost of storing each value in the model parameters being 64 bits. However, because this remains constant with respect to our data set size, as N increases these ratios will approach $\frac{\log_2(P(\{\mathbf{X}^{(n)}\}_{n=1}^{N}|\theta)}{N \cdot D}$ and we'll recover our compression rates from part (f). By increasing k, we see that the ratio increases, and thus our compression rate is worse. This makes sense because we are essentially slowly storing the data into the model. In the extreme example of $k = N$ we can have the parameters for each mixture model as an image in the data. Although in this case, we could store our parameters as binary values instead of 64 bit floats. Our mixture component is uniform because each mixture is equally likely (all images are equally likely) so there is no need to store any values for π. Thus, we would recover the cost of naive encoding our data. Taking into account the model parameters further verifies that the cost of encoding with this approach is much higher than the cost of gzip.

Question 5

(a) The formulae for the ML estimates of $P(s_i = \alpha | s_{i-1} = \beta) = \Psi(\alpha, \beta)$:

$$\Psi(\alpha, \beta) = \frac{N_{\alpha,\beta}}{N_\beta}$$

where $N_{\alpha,\beta}$ is the count of the number of occurrences of the pair (α, β), where β is before α in the text and N_β is the number of occurrences of β. Moreover to ensure ergodicity, a one was added to each $N_{\alpha,\beta}$. This was also taken into account for the normaliser N_β.

Moreover, the stationary distribution ϕ can be calculated using the power method:

(i) Initialise any $\phi^{(0)} \in \mathbb{R}^{53\times1}$ and $\sum_i \phi_i^{(0)} = 1$

(ii) Repeat $\phi^{(i+1)} = \Psi\phi^{(i)}$

(iii) Terminate when $\phi^{(i+1)} - \phi^{(i)} < \epsilon$

where $\Psi \in \mathbf{R}^{53\times53}$ containing the transition probabilities, $\Psi_{i,j} = P(s^j | s^i)$ where s^i is the i^{th} symbol and s^j is the j^{th} symbol with respect to the indices of symbols in the Ψ matrix, and ϵ is some small number indicating sufficient convergence of the distribution to be considered stationary. The function $\phi(\gamma)$ is simply the index of symbol γ in the vector ϕ.

The transition matrix Ψ:

	=	space	-	","	;	:	!	?	/	.	'	double quotes	(	)	[	]	*	0	1	2	3	4	5	6	7	8
=	1.8e-02	5.9e-06	5.3e-04	2.5e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	3.2e-05	1.7e-02	1.3e-02	1.4e-03	1.4e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
space	5.5e-02	4.2e-03	9.0e-03	2.5e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	6.5e-04	3.3e-02	1.5e-01	8.1e-01	1.4e-03	3.6e-02	1.8e-02	7.9e-01	1.3e-02	5.1e-01	1.2e-01	1.3e-01	1.3e-01	6.9e-02	1.7e-01	7.7e-02	3.7e-02
-	1.8e-02	3.3e-05	3.2e-03	2.5e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	3.2e-05	1.7e-02	1.3e-02	1.4e-03	1.4e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	4.5e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	2.8e-02	1.1e-02	8.2e-03
","	1.8e-02	6.7e-02	5.3e-04	2.5e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	6.5e-05	1.7e-02	2.7e-02	4.2e-03	1.4e-03	1.8e-02	1.8e-02	5.9e-03	4.5e-02	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	1.9e-02	1.1e-02	4.1e-03
;	1.8e-02	2.1e-03	5.3e-04	2.5e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	3.2e-05	1.7e-02	1.3e-02	1.4e-03	1.4e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
:	1.8e-02	1.5e-03	5.3e-04	2.5e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	6.5e-05	1.7e-02	1.3e-02	4.2e-03	1.4e-03	1.8e-02	1.8e-02	8.8e-03	4.5e-03	6.8e-03	1.0e-02	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
!	1.8e-02	3.0e-03	5.3e-04	2.5e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	5.7e-03	1.7e-02	1.3e-02	1.4e-03	5.7e-03	1.8e-02	1.8e-02	5.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
?	1.8e-02	2.3e-03	5.3e-04	2.5e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	3.8e-03	1.7e-02	1.3e-02	4.2e-03	4.3e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
/	1.8e-02	2.0e-06	5.3e-04	2.5e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	3.2e-05	1.7e-02	1.3e-02	1.4e-03	1.4e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	2.6e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
.	1.8e-02	2.9e-02	5.3e-04	2.3e-04	8.4e-04	9.4e-04	3.0e-03	5.4e-03	1.6e-02	1.3e-01	1.7e-02	2.7e-02	2.4e-02	1.0e-01	1.8e-02	1.8e-02	1.5e-02	4.5e-03	5.9e-02	2.0e-02	7.1e-02	3.9e-02	2.9e-02	2.8e-02	5.5e-02	2.1e-02
'	1.8e-02	2.0e-06	5.3e-04	5.0e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	3.2e-05	1.7e-02	1.3e-02	1.4e-03	1.4e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
double quotes	1.8e-02	2.0e-05	5.3e-04	2.5e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	3.2e-05	1.7e-02	1.3e-02	1.4e-03	2.9e-03	1.8e-02	1.8e-02	5.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
(	1.8e-02	2.0e-06	5.3e-04	2.5e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	3.2e-05	1.7e-02	2.7e-02	1.4e-03	1.4e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	4.1e-02	2.0e-01	1.4e-01	5.3e-02	2.0e-02	9.4e-03	1.1e-02	8.2e-03
)	1.8e-02	7.8e-04	5.3e-04	4.3e-03	5.1e-03	1.9e-03	2.5e-04	3.2e-04	1.6e-02	1.2e-03	1.7e-02	1.3e-02	2.8e-03	1.4e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
[	1.8e-02	2.0e-06	5.3e-04	2.5e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	3.2e-05	1.7e-02	1.3e-02	1.4e-03	1.4e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
]	1.8e-02	2.0e-06	5.3e-04	2.5e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	3.2e-05	1.7e-02	1.3e-02	1.4e-03	1.4e-03	3.6e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
*	1.8e-02	4.9e-04	5.3e-04	2.5e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	3.2e-05	1.7e-02	1.3e-02	2.4e-02	1.4e-03	1.8e-02	1.8e-02	2.6e-02	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
0	1.8e-02	8.2e-05	5.3e-04	3.0e-04	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	6.5e-05	1.7e-02	1.3e-02	1.4e-03	2.9e-03	1.8e-02	3.6e-02	2.9e-03	1.3e-01	1.1e-02	2.0e-02	8.9e-03	1.3e-02	2.7e-01	1.2e-01	1.8e-01	2.5e-02
1	1.8e-02	3.5e-05	5.3e-04	1.3e-04	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	1.6e-03	1.7e-02	1.3e-02	2.8e-03	2.7e-02	1.8e-02	1.8e-02	2.9e-03	7.2e-02	3.8e-02	3.2e-01	1.9e-01	3.9e-02	9.8e-02	3.8e-02	9.9e-02	6.9e-01
2	1.8e-02	6.7e-05	5.3e-04	3.5e-04	8.4e-04	9.4e-04	2.5e-04	6.3e-04	1.6e-02	4.5e-04	1.7e-02	1.3e-02	1.4e-03	5.7e-02	1.8e-02	3.6e-02	2.9e-03	5.8e-02	6.8e-03	3.1e-02	2.7e-02	5.3e-02	3.9e-02	2.8e-02	3.3e-02	4.1e-03
3	1.8e-02	2.5e-05	5.3e-04	1.8e-04	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	2.6e-04	1.7e-02	1.3e-02	1.4e-03	2.4e-02	1.8e-02	1.8e-02	2.9e-03	3.6e-02	4.5e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
4	1.8e-02	9.8e-06	1.1e-03	7.5e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	2.3e-04	1.7e-02	1.3e-02	1.4e-03	5.7e-03	1.8e-02	1.8e-02	2.9e-03	1.3e-02	6.8e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
5	1.8e-02	1.8e-05	1.1e-03	4.3e-04	1.7e-03	1.9e-03	2.5e-04	3.2e-04	3.2e-02	2.9e-04	1.7e-02	1.3e-02	1.4e-03	2.9e-03	1.8e-02	1.8e-02	2.9e-03	3.1e-02	2.3e-03	5.1e-03	8.9e-03	2.6e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
6	1.8e-02	2.2e-05	1.6e-03	2.3e-04	1.7e-03	9.4e-04	2.5e-04	6.3e-04	1.6e-02	1.9e-04	1.7e-02	1.3e-02	1.4e-03	1.4e-03	1.8e-02	1.8e-02	2.9e-03	2.7e-02	2.3e-03	1.5e-02	8.9e-03	2.6e-02	9.8e-03	1.2e-01	2.2e-02	4.1e-03
7	1.8e-02	2.7e-05	5.3e-04	2.0e-04	8.4e-04	9.4e-04	5.0e-04	3.2e-04	1.6e-02	1.6e-04	1.7e-02	1.3e-02	1.4e-03	1.4e-03	1.8e-02	1.8e-02	2.9e-03	1.8e-02	4.5e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	2.1e-02
8	1.8e-02	1.6e-05	5.3e-04	7.5e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	1.6e-04	1.7e-02	1.3e-02	1.4e-03	1.4e-03	1.8e-02	1.8e-02	2.9e-03	3.6e-01	1.9e-01	2.6e-02	8.9e-03	2.6e-02	9.8e-03	1.9e-02	2.2e-02	8.2e-03
9	1.8e-02	2.7e-05	5.3e-04	1.5e-04	8.4e-04	1.9e-03	2.5e-04	3.2e-04	1.6e-02	1.9e-04	1.7e-02	1.3e-02	1.4e-03	1.4e-03	1.8e-02	1.8e-02	2.9e-03	1.8e-02	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	1.9e-02	2.2e-02	4.1e-03
a	1.8e-02	2.4e-02	2.0e-02	2.4e-02	1.7e-02	6.6e-03	3.3e-02	1.7e-02	1.6e-02	1.6e-02	1.7e-02	1.3e-02	4.2e-03	1.4e-02	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
b	1.8e-02	2.2e-04	3.2e-03	9.0e-04	8.4e-04	9.4e-04	5.0e-04	9.5e-04	1.6e-02	6.5e-04	1.7e-02	1.3e-02	1.4e-03	4.3e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
c	1.8e-02	9.2e-04	1.6e-03	9.8e-04	3.4e-03	1.9e-03	1.0e-03	6.3e-04	1.6e-02	1.0e-03	1.7e-02	1.3e-02	1.4e-03	5.7e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
d	1.8e-02	1.2e-01	1.2e-01	1.1e-01	1.0e-01	1.8e-01	8.0e-02	5.9e-02	1.6e-02	9.1e-02	1.7e-02	2.7e-02	7.1e-03	6.8e-02	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
e	1.8e-02	1.8e-01	1.5e-01	1.5e-01	1.3e-01	1.5e-01	1.7e-01	1.8e-01	1.6e-02	1.4e-01	3.3e-02	1.3e-02	8.5e-03	1.1e-01	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	1.0e-02	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
f	1.8e-02	3.3e-02	9.0e-02	1.2e-02	8.4e-03	1.9e-02	1.3e-02	9.8e-03	1.6e-02	1.4e-02	1.7e-02	1.3e-02	2.8e-03	1.9e-02	1.8e-02	1.8e-02	2.9e-03	4.5e-03	1.1e-02	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
g	1.8e-02	3.5e-02	5.5e-02	3.7e-02	3.7e-02	5.8e-02	3.5e-02	4.4e-02	8.1e-02	3.4e-02	1.7e-02	8.0e-02	1.4e-03	3.1e-02	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
h	1.8e-02	2.4e-02	2.0e-02	3.2e-02	2.6e-02	1.4e-02	5.0e-02	3.5e-02	1.6e-02	1.8e-02	1.7e-02	1.3e-02	5.6e-03	1.7e-02	1.8e-02	1.8e-02	2.9e-03	4.5e-03	4.5e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
i	1.8e-02	8.9e-03	1.2e-02	5.6e-03	3.4e-03	1.9e-03	5.5e-03	1.4e-02	1.6e-02	4.2e-03	1.7e-02	4.0e-02	2.8e-03	1.4e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
j	1.8e-02	2.0e-06	5.3e-04	2.5e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	3.2e-05	1.7e-02	1.3e-02	1.4e-03	1.4e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
k	1.8e-02	7.3e-03	1.8e-02	1.4e-02	1.4e-02	1.0e-02	1.6e-02	1.7e-02	3.2e-02	1.3e-02	1.7e-02	1.3e-02	2.8e-03	1.0e-02	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
l	1.8e-02	1.9e-02	6.0e-02	3.5e-02	2.9e-02	2.1e-02	3.6e-02	2.3e-02	1.6e-02	2.0e-02	1.7e-02	1.3e-02	2.8e-03	2.0e-02	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
m	1.8e-02	1.7e-02	4.3e-03	3.6e-02	5.4e-02	4.3e-02	3.0e-02	3.3e-02	1.6e-02	5.3e-02	3.3e-02	1.3e-02	1.4e-03	3.9e-02	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
n	1.8e-02	6.4e-02	8.3e-02	7.6e-02	7.6e-02	6.6e-02	6.6e-02	6.3e-02	1.6e-02	6.5e-02	6.7e-02	2.7e-02	5.6e-03	5.7e-02	1.8e-02	1.8e-02	2.9e-03	4.5e-03	9.0e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
o	1.8e-02	4.6e-02	4.3e-02	2.1e-02	1.3e-02	1.4e-02	3.6e-02	3.9e-02	1.6e-02	1.2e-02	1.7e-02	1.3e-02	2.8e-03	1.4e-02	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
p	1.8e-02	4.4e-03	4.3e-03	9.6e-03	7.6e-03	5.7e-03	9.8e-03	5.4e-03	1.6e-02	8.8e-03	1.7e-02	1.3e-02	2.8e-03	7.1e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
q	1.8e-02	3.9e-06	5.3e-04	5.0e-05	8.4e-04	9.4e-04	2.5e-04	3.2e-04	1.6e-02	3.2e-05	1.7e-02	1.3e-02	1.4e-03	1.4e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
r	1.8e-02	4.7e-02	9.5e-02	7.1e-02	8.2e-02	5.8e-02	6.6e-02	8.3e-02	1.6e-02	6.8e-02	1.7e-02	1.3e-02	7.1e-03	5.1e-02	1.8e-02	1.8e-02	5.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
s	1.8e-02	9.4e-02	2.7e-02	1.5e-01	1.5e-01	1.3e-01	1.3e-01	8.4e-02	3.2e-02	1.1e-01	3.3e-02	1.3e-02	1.1e-02	1.1e-01	1.8e-02	1.8e-02	5.9e-03	4.5e-03	4.5e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
t	1.8e-02	9.4e-02	5.6e-02	8.1e-02	9.4e-02	5.8e-02	8.9e-02	1.4e-01	1.6e-02	8.0e-02	1.7e-02	1.3e-02	5.6e-03	6.4e-02	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	1.8e-02	1.3e-02	2.0e-02	9.4e-03	1.1e-02	4.1e-03
u	1.8e-02	5.8e-03	2.1e-03	7.8e-03	8.4e-03	5.7e-03	2.7e-02	3.1e-02	1.6e-02	6.4e-03	1.7e-02	1.3e-02	1.4e-03	5.7e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
v	1.8e-02	3.1e-03	6.4e-03	1.4e-02	1.0e-02	6.6e-03	7.3e-03	8.2e-03	1.6e-02	9.1e-03	1.7e-02	1.3e-02	2.8e-03	1.0e-02	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
w	1.8e-02	1.2e-02	1.2e-02	2.4e-02	2.3e-02	1.5e-02	2.3e-02	2.7e-02	1.6e-02	1.7e-02	1.7e-02	1.3e-02	2.8e-03	2.6e-02	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
x	1.8e-02	4.7e-04	4.8e-03	9.5e-04	8.4e-04	3.8e-03	7.6e-04	3.2e-04	3.2e-02	6.5e-04	1.7e-02	1.3e-02	1.4e-03	1.4e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
y	1.8e-02	4.6e-02	8.0e-02	7.9e-02	7.8e-02	9.7e-02	6.7e-02	6.9e-02	3.2e-02	7.0e-02	1.7e-02	1.3e-02	4.2e-03	5.4e-02	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03
z	1.8e-02	1.4e-04	3.2e-03	8.0e-04	1.7e-03	1.9e-03	1.0e-03	9.5e-04	1.6e-02	4.9e-04	1.7e-02	1.3e-02	1.4e-03	2.9e-03	1.8e-02	1.8e-02	2.9e-03	4.5e-03	2.3e-03	5.1e-03	8.9e-03	1.3e-02	9.8e-03	9.4e-03	1.1e-02	4.1e-03

	9	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
=	1.2e-02	5.0e-06	2.9e-05	1.6e-05	8.5e-06	3.2e-06	1.8e-05	2.0e-05	6.0e-06	5.9e-06	3.9e-04	5.0e-05	1.1e-05	1.6e-05	5.5e-06	5.3e-06	2.2e-05	4.2e-04	6.8e-06	6.4e-06	4.5e-06	1.6e-05	4.0e-05	1.7e-05	2.3e-04	2.2e-05	5.5e-04
space	7.1e-02	3.1e-01	6.6e-01	3.1e-01	1.3e-01	3.3e-02	3.5e-01	1.7e-01	2.7e-01	1.6e-01	5.7e-01	1.8e-01	1.3e-01	2.9e-01	7.3e-02	1.7e-01	3.6e-01	5.4e-01	9.2e-02	2.4e-01	3.6e-01	7.9e-02	1.4e-01	6.3e-01	1.0e-01	1.3e-01	7.0e-02
-	1.2e-02	2.5e-04	3.4e-03	2.9e-03	1.3e-03	2.5e-04	2.4e-03	7.1e-04	4.4e-04	5.3e-04	7.7e-04	1.8e-03	1.3e-03	1.1e-03	3.7e-04	5.0e-04	1.2e-03	8.5e-04	3.7e-04	1.1e-03	6.0e-04	4.8e-04	2.8e-04	7.6e-04	2.3e-04	7.3e-04	7.1e-03
","	1.2e-02	3.0e-03	5.8e-03	1.9e-03	6.5e-04	2.3e-04	1.5e-03	1.2e-03	8.6e-04	6.4e-04	4.6e-03	1.0e-03	6.3e-04	1.1e-03	3.4e-04	3.8e-04	2.4e-03	3.4e-03	6.2e-04	1.3e-03	1.1e-03	3.4e-04	5.5e-04	4.7e-03	2.3e-04	5.3e-04	1.1e-03
;	1.2e-02	6.0e-05	2.6e-04	3.3e-05	8.5e-06	9.6e-06	3.7e-05	3.9e-05	6.6e-05	5.3e-05	3.9e-04	5.0e-05	2.1e-05	3.3e-05	1.1e-05	2.6e-05	6.6e-05	4.2e-04	6.8e-06	5.1e-05	9.4e-05	1.6e-05	4.0e-05	6.9e-05	2.3e-04	6.6e-05	5.5e-04
:	1.2e-02	4.0e-05	5.9e-05	3.3e-05	3.4e-05	9.6e-06	3.7e-05	2.0e-05	5.4e-05	4.1e-05	3.9e-04	5.0e-05	2.1e-05	6.5e-05	5.5e-06	1.6e-05	1.1e-04	4.2e-04	6.8e-06	5.1e-05	5.8e-05	1.6e-05	4.0e-05	6.9e-05	2.3e-04	2.2e-05	5.5e-04
!	1.2e-02	6.5e-05	1.2e-04	3.3e-05	2.5e-05	1.3e-05	3.7e-05	5.9e-05	4.2e-05	5.3e-05	7.7e-04	2.5e-04	4.2e-05	9.8e-05	2.8e-05	1.6e-05	4.4e-05	4.2e-04	6.8e-06	2.6e-05	6.3e-05	1.6e-05	1.2e-04	2.3e-04	2.3e-04	1.3e-04	5.5e-04
?	1.2e-02	3.0e-05	1.8e-04	4.9e-05	3.4e-05	9.6e-06	5.5e-05	2.0e-05	8.4e-05	1.1e-04	3.9e-04	5.0e-05	2.1e-05	3.3e-05	4.4e-05	2.1e-05	4.4e-05	4.2e-04	6.8e-06	1.9e-05	7.6e-05	1.6e-05	4.0e-05	2.9e-04	2.3e-04	1.1e-04	5.5e-04
/	1.2e-02	5.0e-06	2.9e-05	4.9e-05	2.5e-05	3.2e-06	1.8e-05	2.0e-05	1.2e-05	5.9e-06	3.9e-04	5.0e-05	2.1e-05	1.6e-05	5.5e-06	5.3e-06	2.2e-05	4.2e-04	6.8e-06	6.4e-06	9.0e-06	1.6e-05	4.0e-05	1.7e-05	2.3e-04	4.4e-05	5.5e-04
.	4.8e-02	3.2e-03	9.4e-03	6.0e-03	1.1e-03	2.7e-04	2.3e-03	5.7e-04	2.3e-03	2.0e-03	1.4e-02	2.6e-03	4.3e-04	1.3e-03	1.5e-03	1.2e-03	8.8e-03	1.7e-03	6.4e-04	1.9e-03	5.2e-03	1.7e-04	2.4e-04	5.2e-03	2.3e-04	5.5e-04	3.8e-03
'	1.2e-02	9.9e-06	2.9e-05	1.6e-05	8.5e-06	3.2e-06	1.8e-05	2.0e-05	6.0e-06	5.9e-06	3.9e-04	5.0e-05	1.1e-05	1.6e-05	5.5e-06	5.3e-06	2.2e-05	4.2e-04	6.8e-06	3.9e-05	4.5e-06	1.6e-05	4.0e-05	1.7e-05	2.3e-04	2.2e-05	5.5e-04
double quotes	1.2e-02	5.0e-06	2.9e-05	1.6e-05	1.7e-05	3.2e-06	1.8e-05	2.0e-05	6.0e-06	1.2e-05	3.9e-04	5.0e-05	1.1e-05	1.6e-05	5.5e-06	5.3e-06	1.8e-04	4.2e-04	1.4e-05	6.4e-06	9.0e-06	1.6e-05	4.0e-05	1.7e-05	2.3e-04	2.2e-05	5.5e-04
(	1.2e-02	4.1e-04	5.0e-04	9.8e-05	6.8e-05	2.6e-05	2.7e-04	9.8e-05	4.6e-04	2.6e-04	7.7e-04	3.0e-04	7.4e-05	2.1e-04	9.9e-05	1.8e-04	5.3e-04	8.5e-04	3.4e-05	2.9e-04	4.5e-04	4.7e-05	1.6e-04	1.4e-03	2.3e-04	4.4e-05	5.5e-04
)	1.2e-02	4.0e-05	8.8e-05	1.6e-05	8.5e-06	3.2e-06	3.7e-05	2.0e-05	1.8e-05	2.3e-05	3.9e-04	5.0e-05	1.1e-05	3.3e-05	5.5e-06	2.1e-05	6.6e-05	4.2e-04	6.8e-06	1.3e-05	3.6e-05	1.6e-05	4.0e-05	6.9e-05	2.3e-04	2.2e-05	5.5e-04
[	1.2e-02	5.0e-06	2.9e-05	1.6e-05	8.5e-06	6.4e-06	1.8e-05	2.0e-05	6.0e-06	5.9e-06	3.9e-04	5.0e-05	1.1e-05	3.3e-05	5.5e-06	5.3e-06	2.2e-05	4.2e-04	6.8e-06	6.4e-06	4.5e-06	1.6e-05	4.0e-05	1.7e-05	2.3e-04	2.2e-05	5.5e-04
]	1.2e-02	5.0e-06	2.9e-05	1.6e-05	8.5e-06	3.2e-06	1.8e-05	2.0e-05	6.0e-06	5.9e-06	3.9e-04	5.0e-05	2.1e-05	1.6e-05	5.5e-06	5.3e-06	2.2e-05	4.2e-04	6.8e-06	6.4e-06	4.5e-06	1.6e-05	4.0e-05	1.7e-05	2.3e-04	2.2e-05	5.5e-04
*	1.2e-02	1.5e-05	2.9e-05	3.3e-05	1.7e-05	3.2e-06	1.8e-05	2.0e-05	6.0e-06	1.2e-05	3.9e-04	1.0e-04	1.1e-05	1.6e-05	5.5e-06	5.3e-06	4.4e-05	4.2e-04	6.8e-06	6.4e-06	4.5e-06	3.1e-05	4.0e-05	5.2e-05	2.3e-04	4.4e-05	5.5e-04
0	2.1e-01	5.0e-06	2.9e-05	4.9e-05	8.5e-06	3.2e-06	1.8e-05	2.0e-05	6.0e-06	5.9e-06	3.9e-04	5.0e-05	1.1e-05	1.6e-05	5.5e-06	5.3e-06	2.2e-05	4.2e-04	6.8e-06	6.4e-06	4.5e-06	1.6e-05	4.0e-05	3.5e-05	2.3e-04	2.2e-05	5.5e-04
1	1.2e-02	5.0e-06	2.9e-05	3.3e-05	8.5e-06	3.2e-06	1.8e-05	2.0e-05	6.0e-06	5.9e-06	3.9e-04	5.0e-05	1.1e-05	1.6e-05	5.5e-06	5.3e-06	2.2e-05	4.2e-04	6.8e-06	1.3e-05	4.5e-06	1.6e-05	4.0e-05	1.7e-05	2.3e-04	2.2e-05	5.5e-04
2	1.2e-02	9.9e-06	2.9e-05	1.3e-04	8.5e-06	3.2e-06	1.8e-05	2.0e-05	6.0e-06	5.9e-06	3.9e-04	1.0e-04	1.1e-05	1.6e-05	1.1e-05	5.3e-06	2.2e-05	4.2e-04	6.8e-06	6.4e-06	4.5e-06	1.6e-05	4.0e-05	1.7e-05	2.3e-04	2.2e-05	5.5e-04
3	1.2e-02	5.0e-06	2.9e-05	3.3e-05	8.5e-06	3.2e-06	1.8e-05	2.0e-05	6.0e-06	5.9e-06	3.9e-04	5.0e-05	1.1e-05	1.6e-05	5.5e-06	5.3e-06	2.2e-05	4.2e-04	4.1e-05	6.4e-06	1.8e-05	1.6e-05	4.0e-05	1.7e-05	2.3e-04	2.2e-05	5.5e-04
4	1.2e-02	5.0e-06	2.9e-05	1.6e-05	8.5e-06	3.2e-06	1.8e-05	2.0e-05	6.0e-06	5.9e-06	3.9e-04	5.0e-05	1.1e-05	1.6e-05	5.5e-06	5.3e-06	2.2e-05	4.2e-04	6.8e-06	6.4e-06	1.3e-05	1.6e-05	4.0e-05	1.7e-05	4.5e-04	2.2e-05	5.5e-04
5	2.4e-02	5.0e-06	2.9e-05	6.5e-05	8.5e-06	3.2e-06	1.8e-05	2.0e-05	6.0e-06	5.9e-06	3.9e-04	5.0e-05	1.1e-05	1.6e-05	5.5e-06	5.3e-06	2.2e-05	4.2e-04	6.8e-06	6.4e-06	9.0e-06	1.6e-05	4.0e-05	1.7e-05	2.3e-04	4.4e-05	5.5e-04
6	1.2e-02	5.0e-06	2.9e-05	3.3e-05	8.5e-06	3.2e-06	1.8e-05	2.0e-05	6.0e-06	5.9e-06	3.9e-04	5.0e-05	1.1e-05	1.6e-05	5.5e-06	5.3e-06	2.2e-05	4.2e-04	6.8e-06	6.4e-06	1.8e-05	1.6e-05	4.0e-05	1.7e-05	2.3e-04	2.2e-05	5.5e-04
7	2.4e-02	5.0e-06	2.9e-05	3.3e-05	8.5e-06	3.2e-06	1.8e-05	2.0e-05	6.0e-06	5.9e-06	3.9e-04	5.0e-05	1.1e-05	1.6e-05	5.5e-06	5.3e-06	2.2e-05	4.2e-04	6.8e-06	6.4e-06	1.8e-05	1.6e-05	4.0e-05	1.7e-05	2.3e-04	2.2e-05	5.5e-04
8	4.8e-02	5.0e-06	2.9e-05	1.6e-05	8.5e-06	3.2e-06	1.8e-05	2.0e-05	6.0e-06	5.9e-06	3.9e-04	5.0e-05	1.1e-05	1.6e-05	5.5e-06	1.1e-05	4.4e-05	4.2e-04	6.8e-06	6.4e-06	1.3e-05	1.6e-05	4.0e-05	1.7e-05	2.3e-04	2.2e-05	5.5e-04
9	1.2e-02	5.0e-06	5.9e-05	1.6e-05	8.5e-06	3.2e-06	1.8e-05	2.0e-05	6.0e-06	5.9e-06	3.9e-04	5.0e-05	1.1e-05	1.6e-05	5.5e-06	5.3e-06	2.2e-05	4.2e-04	6.8e-06	6.4e-06	9.0e-06	1.6e-05	4.0e-05	1.7e-05	2.3e-04	2.2e-05	5.5e-04
a	1.2e-02	1.3e-04	1.0e-01	1.2e-01	9.6e-02	2.4e-04	3.1e-02	6.7e-02	2.1e-03	5.1e-02	6.8e-02	1.3e-01	1.5e-01	7.7e-02	2.5e-01	2.5e-04	1.0e-01	3.8e-03	1.2e-01	1.3e-01	1.3e-01	3.9e-02	1.7e-01	3.8e-02	1.2e-02	1.2e-01	1.9e-01
b	1.2e-02	1.6e-02	6.8e-03	6.5e-05	1.6e-04	3.6e-02	1.8e-05	2.0e-05	1.8e-05	5.9e-03	6.8e-02	5.0e-05	3.8e-02	1.1e-03	5.0e-05	2.2e-02	2.2e-05	4.2e-04	1.3e-02	3.1e-03	1.2e-03	8.2e-02	1.9e-03	3.8e-04	2.3e-04	5.7e-02	5.5e-04
c	1.2e-02	3.3e-02	8.8e-05	1.7e-02	9.3e-05	4.2e-02	7.3e-05	9.8e-05	7.0e-02	1.5e-02	3.9e-04	1.3e-01	2.1e-02	9.8e-05	1.1e-05	6.8e-02	6.6e-05	5.0e-02	1.4e-02	6.8e-04	1.9e-02	2.6e-02	4.0e-05	1.7e-04	2.3e-04	6.9e-03	3.3e-03
d	1.2e-02	1.5e-02	9.1e-03	3.3e-03	1.3e-02	4.4e-02	4.6e-03	1.0e-02	2.7e-03	4.7e-02	1.0e-01	5.1e-03	1.4e-02	7.9e-03	2.9e-03	2.7e-02	5.0e-03	2.6e-02	2.5e-02	1.7e-02	3.7e-03	1.8e-02	1.4e-02	5.2e-03	2.3e-04	2.7e-02	3.3e-03
e	1.2e-02	7.0e-02	1.7e-02	9.6e-02	2.5e-01	2.6e-02	6.5e-02	4.5e-02	7.2e-03	2.2e-02	3.8e-02	1.6e-02	1.1e-01	1.2e-01	1.5e-01	1.1e-02	8.4e-02	1.3e-01	3.0e-01	1.3e-01	3.6e-02	6.2e-03	2.1e-01	6.5e-02	7.1e-01	8.3e-02	9.5e-02
f	1.2e-02	2.0e-02	2.2e-03	9.1e-04	2.4e-04	1.5e-02	5.2e-02	5.9e-04	9.1e-04	2.8e-02	2.3e-03	8.5e-04	1.3e-02	9.2e-04	1.3e-04	4.3e-02	1.1e-03	1.3e-03	3.9e-02	1.4e-03	1.1e-02	2.9e-02	5.9e-04	1.5e-03	2.3e-04	2.3e-03	5.5e-04
g	1.2e-02	1.8e-02	1.8e-03	6.7e-04	7.1e-04	1.9e-02	1.1e-03	9.1e-03	3.7e-02	1.5e-02	2.3e-03	1.5e-04	1.6e-02	9.6e-04	5.3e-03	1.7e-02	1.1e-03	8.5e-04	1.9e-02	6.1e-03	2.0e-03	2.1e-02	5.1e-04	1.8e-03	2.3e-04	2.2e-03	1.6e-03
h	1.2e-02	1.4e-01	2.6e-03	1.7e-03	8.5e-04	2.4e-01	1.7e-03	4.7e-04	7.3e-04	1.5e-01	7.7e-04	5.0e-03	1.7e-03	6.7e-03	7.8e-04	7.2e-02	1.1e-03	4.2e-04	8.9e-03	1.5e-03	2.0e-02	2.5e-02	5.1e-04	1.8e-03	2.3e-04	2.0e-02	5.5e-04
i	1.2e-02	1.5e-02	3.8e-02	1.4e-01	7.4e-02	2.7e-02	6.9e-02	8.4e-02	2.4e-04	2.9e-03	3.9e-04	6.0e-02	8.4e-02	1.6e-01	2.7e-01	3.8e-02	1.9e-02	1.2e-02	4.0e-02	1.4e-01	9.5e-02	2.1e-03	1.4e-01	8.7e-04	7.3e-02	4.4e-05	3.2e-01
j	1.2e-02	4.8e-04	2.9e-05	1.6e-05	8.5e-06	1.8e-03	1.8e-05	2.0e-05	6.0e-06	7.0e-05	3.9e-04	5.0e-05	1.1e-05	1.6e-05	5.5e-06	3.8e-03	2.2e-05	4.2e-04	6.8e-06	6.4e-06	4.5e-06	1.8e-02	4.0e-05	1.7e-05	2.3e-04	2.2e-05	5.5e-04
k	1.2e-02	2.7e-03	5.3e-04	9.3e-04	6.8e-05	1.8e-02	4.7e-04	2.7e-04	5.0e-03	2.0e-02	7.7e-04	5.0e-05	3.6e-03	6.0e-04	1.1e-02	1.5e-03	2.4e-04	4.2e-04	6.8e-04	4.9e-03	2.0e-04	1.0e-02	5.5e-04	1.7e-03	2.3e-04	2.3e-03	5.5e-04
l	1.2e-02	3.8e-02	3.0e-03	1.8e-03	5.9e-02	5.4e-02	4.3e-02	2.1e-03	3.2e-04	5.5e-02	3.9e-04	5.2e-02	1.4e-01	7.0e-03	7.5e-04	4.4e-02	9.3e-03	2.5e-03	3.4e-03	1.1e-02	7.9e-03	2.3e-02	1.7e-02	8.8e-03	2.3e-04	2.2e-01	1.4e-02
m	1.2e-02	4.6e-02	3.3e-02	9.0e-04	8.5e-05	4.8e-02	2.9e-03	2.4e-04	3.2e-04	2.8e-02	1.5e-03	5.0e-04	1.8e-03	2.3e-02	1.3e-03	3.5e-02	7.7e-02	4.2e-04	6.4e-04	1.3e-02	8.7e-04	2.6e-02	2.4e-04	9.2e-04	2.3e-04	5.4e-02	5.5e-04
n	1.2e-02	2.9e-02	7.2e-03	1.5e-01	2.9e-01	4.8e-02	1.8e-02	4.9e-01	2.1e-03	3.4e-02	4.8e-02	7.5e-02	2.3e-02	3.5e-03	1.1e-02	6.7e-02	4.4e-03	1.1e-01	1.0e-03	4.1e-02	7.4e-02	1.5e-02	3.0e-02	4.7e-03	1.9e-02	4.6e-02	1.3e-02
o	1.2e-02	7.5e-03	2.9e-02	2.4e-02	2.7e-02	1.7e-03	3.1e-01	1.6e-02	3.3e-03	1.3e-02	4.8e-02	1.7e-01	7.8e-02	1.8e-01	1.5e-01	3.8e-02	7.0e-02	1.4e-02	1.3e-01	4.4e-02	4.6e-02	3.7e-01	2.3e-01	1.7e-01	3.2e-02	1.6e-02	6.3e-02
p	1.2e-02	2.2e-02	5.9e-04	2.2e-03	4.2e-05	2.6e-02	1.9e-03	5.9e-05	2.6e-03	2.5e-02	7.7e-04	1.0e-03	4.4e-02	5.1e-04	5.5e-05	2.5e-02	5.8e-02	4.2e-04	5.3e-02	6.9e-03	9.9e-03	1.8e-02	7.9e-05	6.8e-04	2.3e-04	7.5e-03	5.5e-04
q	1.2e-02	5.0e-06	2.9e-05	1.6e-05	8.5e-06	3.2e-06	1.8e-05	2.0e-05	6.0e-06	5.9e-06	3.9e-04	5.0e-05	1.1e-05	1.6e-05	5.5e-06	5.3e-06	2.2e-05	4.2e-04	6.8e-06	6.4e-06	4.5e-06	3.6e-02	4.0e-05	1.7e-05	2.3e-04	2.2e-05	5.5e-04
r	2.4e-02	4.1e-02	9.2e-03	2.6e-02	3.8e-02	1.1e-01	1.1e-02	2.8e-02	2.2e-03	7.7e-02	3.1e-03	4.7e-02	1.4e-02	4.9e-02	1.4e-02	7.8e-02	1.4e-02	6.8e-03	3.4e-02	5.3e-02	2.2e-02	4.5e-02	2.9e-02	6.8e-03	2.3e-04	1.3e-01	4.6e-02
s	1.2e-02	4.4e-02	1.7e-02	4.0e-02	2.0e-03	5.6e-02	9.4e-03	2.7e-03	6.6e-02	5.4e-02	7.4e-03	1.1e-01	1.4e-02	3.0e-02	3.1e-03	4.6e-02	7.6e-02	7.8e-02	1.1e-03	6.4e-02	8.1e-02	6.2e-02	4.2e-03	1.7e-02	6.8e-04	8.6e-03	2.7e-03
t	1.2e-02	3.9e-02	4.4e-03	1.3e-02	9.8e-04	6.1e-02	5.6e-03	1.4e-03	4.4e-01	8.3e-02	2.7e-03	2.4e-03	3.2e-02	6.5e-03	1.3e-03	1.3e-01	3.8e-03	5.1e-03	3.9e-02	2.7e-02	2.2e-02	4.7e-02	1.5e-03	2.5e-02	2.3e-04	7.0e-02	7.6e-02
u	1.2e-02	6.6e-03	2.6e-02	3.4e-02	1.2e-02	6.1e-03	6.2e-03	6.4e-02	3.4e-04	1.0e-02	7.7e-04	5.5e-03	6.6e-02	2.2e-02	4.8e-02	8.2e-04	6.2e-02	1.3e-03	5.4e-02	5.7e-02	4.9e-02	6.2e-05	3.2e-03	3.6e-04	9.9e-03	4.6e-04	4.1e-02
v	1.2e-02	8.5e-03	1.8e-04	3.3e-05	2.5e-05	4.6e-02	1.5e-04	2.2e-04	7.8e-05	2.8e-02	1.5e-03	4.5e-04	2.3e-03	9.8e-05	3.0e-03	7.8e-03	1.1e-04	4.2e-04	6.4e-04	1.7e-03	1.2e-04	3.3e-04	7.9e-05	3.8e-04	2.3e-04	2.3e-03	5.5e-04
w	1.2e-02	6.1e-02	4.4e-04	5.7e-04	3.2e-03	2.5e-02	1.2e-03	1.8e-04	7.2e-02	5.9e-02	7.7e-04	3.4e-03	2.2e-03	2.9e-04	1.2e-02	2.4e-02	2.7e-04	8.5e-04	3.6e-03	4.9e-03	3.3e-04	4.0e-04	1.2e-04	1.1e-03	2.3e-04	9.7e-04	5.5e-04
x	1.2e-02	1.7e-03	8.8e-05	9.9e-03	1.7e-05	8.6e-04	7.3e-05	2.0e-05	3.8e-04	3.0e-03	3.9e-04	1.0e-04	3.2e-05	3.3e-05	1.7e-05	7.9e-05	3.0e-02	2.1e-03	2.7e-05	3.2e-05	2.2e-03	3.0e-04	6.2e-03	6.9e-05	3.6e-02	1.1e-04	5.5e-04
y	1.2e-02	7.0e-03	7.3e-03	4.3e-03	9.3e-04	8.4e-03	3.4e-03	1.0e-03	8.1e-04	6.8e-03	4.3e-03	2.5e-03	2.6e-03	3.7e-03	4.5e-04	3.2e-02	2.8e-03	2.1e-03	1.0e-03	9.3e-03	4.4e-03	1.2e-03	5.1e-04	3.8e-03	2.3e-04	2.2e-04	6.6e-03
z	1.2e-02	5.0e-04	2.9e-05	3.3e-05	2.5e-04	2.3e-03	1.8e-05	2.0e-05	9.3e-04	1.4e-03	3.9e-04	2.5e-04	4.7e-04	1.0e-03	9.9e-05	3.3e-03	2.2e-05	4.2e-04	6.8e-06	1.9e-05	4.5e-06	4.2e-04	4.0e-05	6.9e-05	2.3e-04	3.3e-04	2.6e-02

(Apologies for the tiny font, latex was being difficult)

The invariant distribution ϕ:

Symbol	*Probability*
=	1.7e-05
space	1.7e-01
-	6.1e-04
,	1.2e-02
;	3.9e-04
:	2.9e-04
!	6.0e-04
?	4.7e-04
/	1.9e-05
.	7.7e-03
’	1.9e-05
double quotes	2.4e-05
(	2.3e-04
)	2.2e-04
[	1.7e-05
]	1.7e-05
*	1.1e-04
0	6.9e-05
1	1.4e-04
2	6.0e-05
3	3.4e-05
4	2.3e-05
5	3.2e-05
6	3.2e-05
7	2.8e-05
8	7.6e-05
9	2.6e-05
a	6.6e-02
b	1.1e-02
c	2.0e-02
d	3.8e-02
e	1.0e-01
f	1.8e-02
g	1.6e-02
h	5.4e-02
i	5.6e-02
j	8.5e-04
k	6.4e-03
l	3.1e-02
m	2.0e-02
n	5.9e-02
o	6.2e-02
p	1.5e-02
q	7.7e-04
r	4.7e-02
s	5.2e-02
t	7.2e-02
u	2.1e-02
v	8.5e-03
w	1.9e-02
x	1.4e-03
y	1.5e-02
z	7.4e-04

(b) The latent variables $\sigma(s)$ for different symbols s are not independent. This is because by choosing an encoding for one symbol $e = \sigma(s)$, the encoding for a second symbol $\sigma(s')$ cannot be e. We have 53 symbols but only 52 degrees of freedom, because once we have defined the encoding for 52 symbols, the encoding for the 53^{rd} symbol cannot be chosen. Thus, there exists a dependence between $\sigma(s)$ for different symbols s.

The joint probability of the encrypted text $e_1 e_2 \cdots e_n$ given σ:

$$P(e_1, e_2, ..., e_n|\sigma) = \phi(\gamma = \sigma^{-1}(e_1)) \prod_{i=2}^{n} \psi(\alpha = \sigma^{-1}(e_i), \beta = \sigma^{-1}(e_{i-1}))$$

because σ is the encoding function, mapping a symbol s into the encoded symbol e, we require σ^{-1} the decoding function mapping the encoded symbol e back to s.

(c) The proposal probability $S(\sigma \to \sigma')$ depends on the permutations of σ and σ'. Our proposal generating process restricts us to choose a proposal σ' that differs from σ only at *two* spots:

$$\sigma'(s^i) = \sigma(s^j)$$

$$\sigma'(s^j) = \sigma(s^i)$$

for any two symbols s^i and s^j of the 53 possible symbols ($s^i \neq s^j$).

Therefore, if the above doesn't hold for σ', $S(\sigma \to \sigma') = 0$. From σ there are $\binom{53}{2}$ possible proposal σ''s with the above property. Because we are assuming a uniform prior distribution over σ's, the transition probability of a σ' that satisfies the above property is $S(\sigma \to \sigma') = \frac{1}{\binom{53}{2}}$.

The MH acceptance probability is given as:

$$A(\sigma \to \sigma'|\mathcal{D}) = \min\{1, \frac{S(\sigma' \to \sigma)P(\sigma'|\mathcal{D})}{S(\sigma \to \sigma')P(\sigma|\mathcal{D})})\}$$

where $S(\sigma \to \sigma')$ is the conditional transition probability of σ' given σ and $\mathcal{D}$ is our encrypted text $e_1, e_2, ..., e_n$.

$S(\sigma \to \sigma') = S(\sigma' \to \sigma)$ for all σ and σ' that differ only at two spots because the probability in this case will always be $\frac{1}{\binom{53}{2}}$, so we can simplify:

$$A(\sigma \to \sigma'|\mathcal{D}) = \min\{1, \frac{P(\sigma'|\mathcal{D})}{P(\sigma|\mathcal{D})})\}$$

From Bayes' Theorem:

$$P(\sigma|\mathcal{D}) = \frac{P(\mathcal{D}|\sigma)P(\sigma)}{\sum_{\sigma'} P(\mathcal{D}|\sigma')P(\sigma')}$$

We are assuming a uniform prior for σ, so $P(\sigma)$ is a constant and we can simplify further:

$$A(\sigma \to \sigma'|\mathcal{D}) = \min\{1, \frac{P(\mathcal{D}|\sigma')}{P(\mathcal{D}|\sigma)})\}$$

This is the acceptance probability for a given proposal σ'. The expression for $P(\mathcal{D}|\sigma)$ is $P(e_1, e_2, ..., e_n|\sigma)$ described in the previous part.

(d) Reporting the current decryption of the first 60 symbols after every 100 iterations:

MH Iteration	Current Decryption
0	6m p2 2namr'=)mk pn=' batm'=)3t' 2')=q p2 8)*9'= r)b' p' qn
100	er pl losrua= drk po=a bstra=dita lad=n pl -df:a= udba pa no
200	er nl loiruah drw noha bitrahdsta ladhp nl xdymah udba na po
300	er nl loiruav srw nova bitravsdta lasvp nl xsymav usba na po
400	er vd dsir,an orw vsna bitranolta daony vd uophan ,oba va ys
500	er c, ,sirdan or. csna bitranolta ,aony c, uophan doba ca ys
600	en ck kyindar on. cyra bitnarolta kaors ck uophar doba ca sy
700	en pk klindar on. plra bitnaroyta kaors pk uochar doba pa sl
800	en p, ,londar in. plra botnariyta ,airs p, fichar diba pa sl
900	en pu ulondar in. plra botnariyta uairs pu fichar diba pa sl
1000	en pl luondar in. pura botnariyta lairs pl fighar diba pa su
1100	en pl luondar in. pura cotnarixta lairs pl fighar dica pa su
1200	en pk kuondar inl pura comnarixma kairs pk fighar dica pa su
1300	en ck kuondar inl cura pomnarixma kairs ck fighar dipa ca su
1400	en ck koundar inl cora pumnarixma kairs ck fighar dipa ca so
1500	en ck koundar inl cora vumnarixma kairs ck fithar diva ca so
1600	en ck koundar inl cora vumnarixma kairs ck fithar diva ca so
1700	an ck kounder inl core vumnerixme keirs ck fither dive ce so
1800	an ck kounler ind core vumnerixme keirs ck fither live ce so
1900	an ck kounler ind core vumnerixme keirs ck fither live ce so
2000	an ck kounler ind core vumnerixme keirs ck fither live ce so
2100	an ck kounler ind core vumnerixme keirs ck fither live ce so
2200	an ck kounler ind core vumnerixme keirs ck fither live ce so
2300	an ck kounger ind core vumnerixme keirs ck fither give ce so
2400	an ck kounger ind core vulnerixle keirs ck fither give ce so
2500	an mk kounger ind more vulnerixle keirs mk fither give me so
2600	an mk kounger ind more vulneriple keirs mk fither give me so
2700	an mk kounger ind more vulneriple keirs mk fither give me so
2800	an mk kounger ind more vulneriple keirs mk fither give me so
2900	an mk kounger ind more vulneriple keirs mk fither give me so
3000	an mk kounger ind more vulneriple keirs mk fither give me so
3100	an mf founger ind more vulneriple feirs mf kither give me so
3200	an mf founger ind more vulneriple feirs mf kither give me so
3300	an mf founger ind more vulneriple feirs mf kither give me so
3400	an mf founger ind more vulneriple feirs mf kither give me so
3500	an mf founger ind more vulneriple feirs mf kither give me so
3600	an mf founger ind more vulneriple feirs mf kither give me so
3700	an mf founger ind more vulneriple feirs mf kither give me so
3800	an mf founger ind more vulneriple feirs mf kither give me so
3900	in mf founger and more vulneraple fears mf kather gave me so
4000	in mf founger and more vulneraple fears mf kather gave me so
4100	in mf founger and more vulneraple fears mf kather gave me so
4200	in mf founger and more vulnerable fears mf kather gave me so
4300	in mf founger and more vulnerable fears mf kather gave me so
4400	in mf founger and more vulnerable fears mf yather gave me so
4500	in mf founger and more vulnerable fears mf yather gave me so
4600	in mf founger and more vulnerable fears mf yather gave me so
4700	in mf founger and more vulnerable fears mf yather gave me so
4800	in mf founger and more vulnerable fears mf yather gave me so
4900	in mf founger and more vulnerable fears mf yather gave me so
5000	in mf founger and more vulnerable fears mf yather gave me so
5100	in mf founger and more vulnerable fears mf yather gave me so
5200	in mf founger and more vulnerable fears mf yather gave me so
5300	in my younger and more vulnerable years my father gave me so
5400	in my younger and more vulnerable years my father gave me so
5500	in my younger and more vulnerable years my father gave me so
5600	in my younger and more vulnerable years my father gave me so
5700	in my younger and more vulnerable years my father gave me so
5800	in my younger and more vulnerable years my father gave me so
5900	in my younger and more vulnerable years my father gave me so
6000	in my younger and more vulnerable years my father gave me so
6100	in my younger and more vulnerable years my father gave me so
6200	in my younger and more vulnerable years my father gave me so
6300	in my younger and more vulnerable years my father gave me so
6400	in my younger and more vulnerable years my father gave me so
6500	in my younger and more vulnerable years my father gave me so
6600	in my younger and more vulnerable years my father gave me so
6700	in my younger and more vulnerable years my father gave me so
6800	in my younger and more vulnerable years my father gave me so
6900	in my younger and more vulnerable years my father gave me so
7000	in my younger and more vulnerable years my father gave me so
7100	in my younger and more vulnerable years my father gave me so
7200	in my younger and more vulnerable years my father gave me so
7300	in my younger and more vulnerable years my father gave me so
7400	in my younger and more vulnerable years my father gave me so
7500	in my younger and more vulnerable years my father gave me so
7600	in my younger and more vulnerable years my father gave me so
7700	in my younger and more vulnerable years my father gave me so
7800	in my younger and more vulnerable years my father gave me so
7900	in my younger and more vulnerable years my father gave me so
8000	in my younger and more vulnerable years my father gave me so
8100	in my younger and more vulnerable years my father gave me so
8200	in my younger and more vulnerable years my father gave me so
8300	in my younger and more vulnerable years my father gave me so
8400	in my younger and more vulnerable years my father gave me so
8500	in my younger and more vulnerable years my father gave me so
8600	in my younger and more vulnerable years my father gave me so
8700	in my younger and more vulnerable years my father gave me so
8800	in my younger and more vulnerable years my father gave me so
8900	in my younger and more vulnerable years my father gave me so
9000	in my younger and more vulnerable years my father gave me so
9100	in my younger and more vulnerable years my father gave me so
9200	in my younger and more vulnerable years my father gave me so
9300	in my younger and more vulnerable years my father gave me so
9400	in my younger and more vulnerable years my father gave me so
9500	in my younger and more vulnerable years my father gave me so
9600	in my younger and more vulnerable years my father gave me so
9700	in my younger and more vulnerable years my father gave me so
9800	in my younger and more vulnerable years my father gave me so
9900	in my younger and more vulnerable years my father gave me so
10000	in my younger and more vulnerable years my father gave me so

The corresponding σ:

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

To help with chain initialisation, 10000 different σ's were first randomly and independently sampled. The σ with the best log-likelihood was chosen as the starting point for the MH chain and the algorithm was then run for 10000 iterations. Moreover, ten different trials of this was performed, where the trial with the best log-likelihood was displayed. The decrypted message for each of the ten trials:

Trial	**Decryption**
0	itedcecoutl featpedof eyunt fa.n ec afredcevas, felay ed ero
1	in my younger and more vulnerable years my father gave me so
2	in cy yomnker and core vmlnerable years cy father kave ce so
3	is hy ytoswer asd htre volseraule yearm hy fanger wave he mt
4	in my younger and more vulnerable years my father gave me so
5	”5407””0””][4)81094307]180([’4819*’80””891207””0:96=810)9(807802]”
6	”542)(2(]94””18234=2)]812:9’4183*’12(13862)(2[307182””3:12)126]”
7	ioadcaclyon earowadle agy.o erk. ac retadcafrsu eanrg ad atl
8	in my younker and more vulnerable years my father kave me so
9	in my younker and more vulnerable years my father kave me so

The Python code for the MH sampler:

```
from typing import Dict, List, Tuple

import numpy as np
import pandas as pd
from sklearn.preprocessing import normalize

from src.constants import DEFAULT_SEED


def _convert_to_scientific_notation(x: float) -> str:
    """
    Convert value to string in scientific notation
    :param x: value to convert
    :return: string of x in scientific notation
    """
    return "{:.1e}".format(float(x))


class Decrypter:
    def __init__(self, decryption_dict: Dict[str, str]) -> None:
        """
        Decrypter containing the mapping a symbol to its encrypted symbol
        :param decryption_dict:
        """
        self.decryption_dict = decryption_dict

    def decrypt(self, encrypted_message: str) -> str:
        """
        Decrypts an encrypted message using the decryption dictionary
        :param encrypted_message: the encrypted message to decrypt
        :return: decrypted message
        """
        return "".join([self.decryption_dict[x] for x in encrypted_message])

    @property
    def table(self) -> pd.DataFrame:
        """
        Generate table containing symbol decryptions
        :return: pandas table of decryptions
        """
        decrpyter_table = pd.DataFrame(
            self.decryption_dict.items(), columns=["s", "sigma(s)"]
        )
        decrpyter_table[decrpyter_table == " "] = "space"
        decrpyter_table[decrpyter_table == '"'] = "double quotes"
        return decrpyter_table.set_index("s")


class Statistics:
    def __init__(
        self,
        training_text: str,
        symbols: List[str],
        invariant_stopping_epsilon: float = 5e-20,
    ) -> None:
        """
        Statistics for text
        :param training_text: training text for calculating transition and invariant probability
        :param symbols: symbols in the training text
        :param invariant_stopping_epsilon: stopping condition for constructing the invariant distribution
        """
        self.training_text = training_text
        self.symbols = symbols
        self.num_symbols = len(symbols)
        self.symbols_dict = self._construct_symbols_dictionary(symbols)
        self.transition_matrix = self._construct_transition_matrix(
            training_text, self.symbols_dict
        )
        self.invariant_distribution = self._approximate_invariant_distribution(
            invariant_stopping_epsilon
        )
        self.log_transition_matrix = np.log(self.transition_matrix)
        self.log_invariant_distribution = np.log(self.invariant_distribution)

    @property
    def list_of_symbols_for_df(self) -> List[str]:
        """
        Replace certain symbols to prepare for dataframe
        :return: list of symbols with some replacements
        """
        x = self.symbols.copy()
        x[x.index(" ")] = "space"
        x[x.index('"')] = "double quotes"
        return x

    @property
    def transition_table(self) -> pd.DataFrame:
        """
        Generate a table containing transition probabilities
        :return: transition probabilities
        """
        df_transitions = pd.DataFrame(
            data=self.transition_matrix,
            columns=self.list_of_symbols_for_df,
```

```python
        )
        df_transitions.index = self.list_of_symbols_for_df
        return df_transitions.applymap(_convert_to_scientific_notation)

    @property
    def invariant_distribution_table(self) -> pd.DataFrame:
        """
        Generate a table containing invariant distribution probabilities
        :return: invariant distribution probabilities
        """
        df = (
            pd.DataFrame(
                data=self.invariant_distribution.reshape(1, -1),
                columns=self.list_of_symbols_for_df,
            )
            .applymap(_convert_to_scientific_notation)
            .transpose()
            .reset_index()
        )
        df.columns = ["Symbol", "Probability"]
        return df.set_index("Symbol")

    @staticmethod
    def _construct_symbols_dictionary(symbols: List[str]) -> Dict[str, int]:
        """
        Construct a dictionary mapping each symbol to an integer to index the transition matrix
        and the invariant distribution
        :param symbols: list of symbols to map
        :return: symbol to integer mapping
        """
        return {k: v for v, k in enumerate(symbols)}

    def _construct_transition_matrix(
        self, text: str, symbols_dict: Dict[str, int]
    ) -> np.ndarray:
        """
        Constructs the transition matrix for a given text
        :param text: string to calculate transition matrix with
        :param symbols_dict: dictionary mapping symbol to a dictionary
        :return:
        """
        # initialise with ones to ensure ergodicity
        transition_matrix = np.ones((self.num_symbols, self.num_symbols))
        for i in range(1, len(text)):
            # check symbols are valid
            if text[i] in symbols_dict and text[i - 1] in symbols_dict:
                transition_matrix[symbols_dict[text[i - 1]], symbols_dict[text[i]]] += 1
        # normalise to get transition probabilities
        transition_matrix = normalize(transition_matrix, axis=0, norm="l1")
        return transition_matrix

    def _approximate_invariant_distribution(
        self, invariant_stopping_epsilon: float
    ) -> np.ndarray:
        """
        Approximate the invariant distribution with the power method
        :param invariant_stopping_epsilon: stopping condition for constructing the invariant distribution
        :return: the invariant distribution as a vector (number of symbols, 1)
        """
        invariant_distribution = np.zeros((self.num_symbols, 1))
        previous_invariant_distribution = invariant_distribution.copy()

        # make sure it's a proper distribution that sums to one
        invariant_distribution[0] = 1

        while (
            np.linalg.norm(invariant_distribution - previous_invariant_distribution)
            > invariant_stopping_epsilon
        ):
            previous_invariant_distribution = invariant_distribution.copy()
            invariant_distribution = self.transition_matrix @ invariant_distribution
        return invariant_distribution

    def log_transition_probability(self, alpha: str, beta: str) -> float:
        """
        Look up the log probability of the transition from symbol alpha to beta
        :param alpha: symbol that is being transitioned from
        :param beta: symbol that is being transitioned to
        :return: probability of transition
        """
        return self.log_transition_matrix[
            self.symbols_dict[beta], self.symbols_dict[alpha]
        ]

    def log_invariant_probability(self, gamma: str) -> float:
        """
        Look up the log probability of a symbol with respect to the invariant distribution
        :param gamma: symbol to query
        :return: log probability of the symbol
        """
        return self.log_invariant_distribution[self.symbols_dict[gamma]].item()

    def compute_log_probability(self, text: str) -> float:
        """
        Compute the log probability of a given text containing symbols
        :param text: text to compute log probability for
```

```python
        :return: log probability of the text
        """
        log_probability = self.log_invariant_probability(text[0])
        for i in range(1, len(text)):
            log_probability += self.log_transition_probability(text[i], text[i - 1])
        return log_probability


class MetropolisHastingsDecryption:
    def __init__(self, symbols: List[str]):
        """
        Metropolis Hastings MCMC for Decryption
        :param symbols: set of symbols to decrypt
        """
        self.symbols = symbols

    def generate_random_decrypter(self) -> Decrypter:
        """
        Generates a random decrypter
        :return: a Decrypter instantiation
        """
        return Decrypter(
            {
                self.symbols[i]: self.symbols[x]
                for i, x in enumerate(
                    np.random.permutation(np.arange(len(self.symbols)))
                )
            }
        )

    @staticmethod
    def generate_proposal_decryption(decrypter: Decrypter) -> Decrypter:
        """
        Generate a proposal decrypter by randomly swapping two of the decryption mappings
        :param decrypter: the decrypter used to generate the proposal
        :return: a proposal decrypter
        """
        x1 = np.random.choice(list(decrypter.decryption_dict.keys()))
        x2 = np.random.choice(list(decrypter.decryption_dict.keys()))
        proposal_decryption = decrypter.decryption_dict.copy()
        proposal_decryption[x2], proposal_decryption[x1] = (
            decrypter.decryption_dict[x1],
            decrypter.decryption_dict[x2],
        )
        return Decrypter(proposal_decryption)

    @staticmethod
    def _choose_decrypter(
        statistics: Statistics,
        encrypted_message: str,
        current_decrypter: Decrypter,
        proposal_decrypter: Decrypter,
    ) -> Decrypter:
        """
        Choose between the current and proposal decrypter
        :param statistics: Statistics instantiation for calculating log probabilities
        :param encrypted_message: the encrypted message
        :param current_decrypter: the current decrypter
        :param proposal_decrypter: the proposal decrypter
        :return:
        """
        # calculate log probabilities
        current_log_probability = statistics.compute_log_probability(
            text=current_decrypter.decrypt(encrypted_message),
        )
        proposal_log_probability = statistics.compute_log_probability(
            text=proposal_decrypter.decrypt(encrypted_message),
        )

        # calculate acceptance probability
        acceptance_probability = np.min(
            [1, np.exp(proposal_log_probability - current_log_probability)]
        )
        # choose decrypter using the acceptance probability
        return np.random.choice(
            [current_decrypter, proposal_decrypter],
            p=[1 - acceptance_probability, acceptance_probability],
        )

    def _find_good_starting_decrypter(
        self,
        statistics: Statistics,
        encrypted_message: str,
        number_start_attempts: int,
    ) -> Decrypter:
        """
        Find a good starting decrypter for the sampler by choosing the one with the best log likelihood
        :param statistics: Statistics instantiation for calculating log probabilities
        :param encrypted_message: the encrypted message
        :param number_start_attempts: number of possible starting decrypters to check
        :return: the best starting decrypter for the sampler
        """
        best_log_likelihood = -np.float("inf")
        best_decrypter = None
        for _ in range(number_start_attempts):
            decrypter = self.generate_random_decrypter()
```

```python
            if (
                statistics.compute_log_probability(
                    text=decrypter.decrypt(encrypted_message)
                )
                > best_log_likelihood
            ):
                best_decrypter = decrypter
        return best_decrypter

    def run(
        self,
        encrypted_message: str,
        statistics: Statistics,
        number_of_mh_loops: int,
        number_start_attempts: int,
        log_decryption_interval: int,
        log_decryption_size: int,
    ) -> Tuple[Decrypter, List[str]]:
        """
        Run the sampler with two steps:
            1. find a good starting decrypter for the sampler
            2. run the sampler
        :param encrypted_message: the encrypted message
        :param statistics: Statistics instantiation for calculating log probabilities
        :param number_of_mh_loops: number of loops to run the metropolis hastings sampler
        :param number_start_attempts: number of possible starting decrypters to check
        :param log_decryption_interval: number of samples between logging the decrypted message
        :param log_decryption_size: number of symbols to decrypt when logging the decrypted message
        :return: a tuple containing the decrypter found from the sampler and the logged decryption message
        """
        decrypter = self._find_good_starting_decrypter(
            statistics, encrypted_message, number_start_attempts
        )
        logged_decryption_message = [
            decrypter.decrypt(encrypted_message)[:log_decryption_size]
        ]
        for i in range(1, number_of_mh_loops + 1):
            if (i + 1) % log_decryption_interval == 0:
                logged_decryption_message.append(
                    decrypter.decrypt(encrypted_message)[:log_decryption_size]
                )
            proposal_decrypter = self.generate_proposal_decryption(decrypter)
            decrypter = self._choose_decrypter(
                statistics, encrypted_message, decrypter, proposal_decrypter
            )
        return decrypter, logged_decryption_message


def _construct_decryptions_table(
    decryption_messages: List[str], decryption_interval: int, columns: List[str]
) -> pd.DataFrame:
    decrypted_message_iterations_table = pd.DataFrame(
        [
            np.arange(0, len(decryption_messages)) * decryption_interval,
            decryption_messages,
        ]
    ).transpose()
    decrypted_message_iterations_table.columns = columns
    return decrypted_message_iterations_table.set_index(columns[0])


def a(
    symbols: List[str],
    training_text: str,
    transition_matrix_path: str,
    invariant_distribution_path: str,
) -> None:
    """
    Produces answers for question 5a
    :param symbols: symbols in the training text
    :param training_text: training text for calculating transition and invariant probability
    :param transition_matrix_path: path to store transition matrix
    :param invariant_distribution_path: path to store invariant distribution
    :return:
    """
    statistics = Statistics(
        training_text,
        symbols,
    )
    statistics.transition_table.to_csv(transition_matrix_path)
    statistics.invariant_distribution_table.to_csv(invariant_distribution_path, sep="|")


def d(
    encrypted_message: str,
    symbols: List[str],
    training_text: str,
    number_trials: int,
    number_of_mh_loops: int,
    number_start_attempts: int,
    log_decryption_interval: int,
    log_decryption_size: int,
    trial_decryptions_table_path: str,
    decryptor_table_path: str,
    decrypted_message_iterations_table_path: str,
) -> None:
```

```python
    """
    Produces answers for question 5d
    :param encrypted_message: the encrypted message
    :param symbols: symbols in the training text
    :param training_text: training text for calculating transition and invariant probability
    :param number_trials: number of times to restart and run the sampler
    :param number_of_mh_loops: number of loops to run the metropolis hastings sampler
    :param number_start_attempts: number of possible starting decrypters to check
    :param log_decryption_interval: number of samples between logging the decrypted message
    :param log_decryption_size: number of symbols to decrypt when logging the decrypted message
    :param trial_decryptions_table_path: path to store decryption messages for each trial
    :param decryptor_table_path: path to store decrypter mapping table
    :param decrypted_message_iterations_table_path: path to store logged decryption messages
    :return:
    """
    statistics = Statistics(
        training_text,
        symbols,
    )
    np.random.seed(DEFAULT_SEED)
    metropolis_hastings_decryption = MetropolisHastingsDecryption(symbols)
    decrypters: List[Decrypter] = []
    log_likelihoods: List[float] = []
    logged_decryption_messages: List[List[str]] = []
    decryption_messages = []
    for i in range(number_trials):
        (decrypter, logged_decryption_message,) = metropolis_hastings_decryption.run(
            encrypted_message,
            statistics,
            number_of_mh_loops,
            number_start_attempts,
            log_decryption_interval,
            log_decryption_size,
        )
        decrypters.append(decrypter)
        log_likelihoods.append(
            statistics.compute_log_probability(decrypter.decrypt(encrypted_message))
        )
        logged_decryption_messages.append(logged_decryption_message)
        decryption_messages.append(
            decrypter.decrypt(encrypted_message)[:log_decryption_size]
        )
    df_trial_decryptions = _construct_decryptions_table(
        decryption_messages=[x[:log_decryption_size] for x in decryption_messages],
        decryption_interval=1,
        columns=["Trial", "Decryption"],
    )
    df_trial_decryptions.to_csv(trial_decryptions_table_path, sep="|")

    # sort trials by log likelihood
    best_trial = np.argmax(log_likelihoods)
    decrypters[best_trial].table.to_csv(decryptor_table_path, sep="|")
    df_logged_decryptions = _construct_decryptions_table(
        decryption_messages=logged_decryption_messages[best_trial],
        decryption_interval=log_decryption_interval,
        columns=["MH Iteration", "Current Decryption"],
    )
    df_logged_decryptions.to_csv(decrypted_message_iterations_table_path, sep="|")
```

src/solutions/q5.py

(e) When some values of $\Psi(\alpha, \beta) = 0$, this affects the ergodicity of the chain. An ergodic chain is one that is irreducible (i.e. all possible transitions between symbols, including to itself, have probability greater than zero). If $\Psi(\alpha, \beta) = 0$, this means that there is zero probability that β will transition to α, breaking our definition. To restore ergodicity, we can add a small transition probability between all symbols of the chain. This essentially acts as a prior, stating that the probability of a symbol to transition to any other symbol (including itself) should never be zero.

(f) If we were to use symbol probabilities alone for decoding, the joint probability would be:

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

the product of the likelihoods of the decoded letters. In this case, the optimal decoding would simply replace the most frequent symbols in the encrypted message with the most frequent symbols in the training text. This decoding approach is much more difficult because each letter is assumed to be independent of its neighbours. For a first order Markov chain, we exploit the structure of language by considering pairs of letters. Assuming that as the training text size approaches infinity and the size of the encrypted message also approaches infinity, that the two will have the same symbol frequency and that the probability of each symbol is unique, (i.e. two different symbols can't have the same frequency), then using symbol probabilities alone should theoretically work by matching symbol probabilities. However, in practise it would be unlikely to be able to make these assumptions about symbol frequencies, especially with the finite size of our training set and encrypted message. Therefore in practise, symbol probabilities alone would not be sufficient.

A second-order chain should also work in theory. However, with this approach it is probably practically more difficult for finding a suitable decoding. This is because our transition tensor would contain N^3 elements, where N is the number of symbols, to account for all possible second order transitions. Our training text would need to increase quadratically to maintain the same ratio of possible transitions to example transitions (number of first order transitions in a text of length N is $N-1$ and second order its $N-2$). This can also introduce sparsity (in this case, small non-zero probabilities because ergodicity is maintained) in our transition tensor. Thus, the log-likelihood of many areas of σ space might be very small or the same as their neighbours, when the transition probabilities are mostly just the offset probability added to maintain ergodicity. Navigating this space will be much more difficult for the sampler.

For an encryption scheme where two symbols map to the same encrypted value:

$$\exists \alpha, \beta, \sigma(\alpha) = \sigma(\beta), \alpha \neq \beta$$

this approach can become much more complicated. Our σ is no longer as easily inverted and therefore for each duplicate mapping, we would have to integrate out the probability for the two possible decrypted symbols when computing the log-likelihood. Moreover, generating proposal encodings is not as simple as swapping the encryption for two symbols. This is because we do not know which two symbols map to the same encrypted symbol and simply swapping would preserve the same collision mapping of the current encoding. Moreover, the number of proposal σ''s will depend on how many duplicates exist in the current σ.

Thus $S(\sigma \to \sigma')$ would no longer be symmetric, complicating the acceptance probability calculation as it would be dependent on the σ and σ'. Overall, this approach could work but would require many changes to accommodate for these complications. Integrating out collision mappings in the log-likelihood, non-symmetric proposal probabilities, and a much larger σ space because duplicates are allowed, means that it will take much longer for the sampler to find a reasonable σ.

If we used this approach for Chinese with ≥ 10000 symbols, we would be attempting to solve the same problem but with $N \geq 10000$ instead of $N = 53$. Similar to the second order Markov chain, although this is theoretically possible, it would require a transition matrix of size $\geq 10000^2$ which is quite impractical and we'd run into similar problems as for second order Markov Chains. An alternative set up could be with using Chinese phonetics, for which there are much fewer than 10000, however this would require a mapping from a phonetic to an encrypted phonetic.

Question 7

(a) To find the local extrema of the function $f(x, y) = x+2y$ subject to the constraint $y^2+xy = 1$, first we define $g(x, y)$:

$$g(x, y) = y^2 + xy - 1$$

where $g(x, y) = 0$ is an equivalent representation of the given constraint.

We can therefore construct the optimisation problem:

$$\min_{\mathbf{x}} f(\mathbf{x})$$

such that $g(\mathbf{x}) = 0$ and $\mathbf{x} := [x, y]^T$.

We can calculate $\nabla f(\mathbf{x})$:

$$\nabla f(\mathbf{x}) = [\frac{\partial}{\partial x}(x + 2y), \frac{\partial}{\partial y}(x + 2y)]^T$$

$$\nabla f(\mathbf{x}) = [1, 2]^T$$

and calculating $\nabla g(\mathbf{x})$:

$$\nabla g(\mathbf{x}) = [\frac{\partial}{\partial x}(y^2 + xy - 1), \frac{\partial}{\partial y}(y^2 + xy - 1)]^T$$

$$\nabla g(\mathbf{x}) = [y, 2y + x]^T$$

Solving the constraint optimisation problem with Lagrange multipliers, we set up the equations:

$$\nabla f(\mathbf{x}) + \lambda \nabla g(\mathbf{x}) = \mathbf{0}$$

and

$$g(\mathbf{x}) = 0$$

Giving us the three equations:

$$1 + \lambda y = 0$$

$$2 + \lambda(2y + x) = 0$$

$$y^2 + xy - 1 = 0$$

Substituting $y = \frac{-1}{\lambda}$ from the first equation into the second equation:

$$2 + \lambda(2(\frac{-1}{\lambda}) + x) = 0$$

$$x = 0$$

Solving for y in our third equation with $x = 0$:

$$y^2 - 1 = 0$$

We see that $y = \pm 1$ and from the first equation $\lambda \mp 1$.

The local extrema are $(x = 0, y = 1)$ when $\lambda = -1$ and $(x = 0, y = -1)$ when $\lambda = 1$.

(b)

(i) Given that $g(a) = \ln(a)$, we want to transform this to the form $f(x, a) = 0$ where $x = g(a)$:

$$x = \ln(a)$$

$$\exp(x) - a = 0$$

Thus,

$$f(x, a) = \exp(x) - a$$

(ii) We know that for Newton's method's

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

where $f(x_n) = \exp(x_n) - a$

We can calculate:

$$f'(x) = \frac{\partial f(x, a)}{\partial x} = \exp(x)$$

Assuming we can evaluate $\exp(x)$, our update equation is:

$$x_{n+1} = x_n - \frac{\exp(x_n) - a}{\exp(x_n)}$$

Simplifying:

$$x_{n+1} = x_n + \frac{a}{\exp(x_n)} - 1$$

we have our update equation in Newton's algorithm for this problem.

Question 8

(a) For:

$$\sup_{\{\mathbf{x}\in\mathbb{R}^n\}} R_A(\mathbf{x})$$

where $R_A(\mathbf{x}) = \frac{\mathbf{x}^T\mathbf{A}\mathbf{x}}{\|\mathbf{x}\|^2}$, we want to show that a maximum is attained.

To do this, we will first show that the above optimisation can be equivalently formulated as:

$$\sup_{\left\{\mathbf{x}\in\mathbb{R}^n \middle| \|\mathbf{x}\|=1\right\}} R_A(\mathbf{x})$$

We begin by considering any $\mathbf{w} \in \mathbb{R}^n$ and let $\mathbf{x} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$. Because $\|\mathbf{x}\| = 1$ we can substitute:

$$\sup_{\left\{\mathbf{x}\in\mathbb{R}^n \middle| \|\mathbf{x}\|=1\right\}} R_A(\mathbf{x}) = \sup_{\left\{\frac{\mathbf{w}}{\|\mathbf{w}\|}\in\mathbb{R}^n \middle| \|\frac{\mathbf{w}}{\|\mathbf{w}\|}\|=1\right\}} \frac{\mathbf{w}^T\mathbf{A}\mathbf{w}\|\mathbf{w}\|^2}{\|\mathbf{w}\|^2\mathbf{w}^T\mathbf{w}}$$

where $\|\mathbf{x}\|^2 = \mathbf{x}^T\mathbf{x}$.

The set $\left\{\frac{\mathbf{w}}{\|\mathbf{w}\|} \in \mathbb{R}^n \middle| \|\frac{\mathbf{w}}{\|\mathbf{w}\|}\| = 1\right\}$ contains all $\mathbf{w} \in \mathbb{R}^n$ so we can rewrite:

$$\sup_{\left\{\mathbf{x}\in\mathbb{R}^n \middle| \|\mathbf{x}\|=1\right\}} R_A(\mathbf{x}) = \sup_{\{\mathbf{w}\in\mathbb{R}^n\}} \frac{\mathbf{w}^T\mathbf{A}\mathbf{w}\|\mathbf{w}\|^2}{\|\mathbf{w}\|^2\mathbf{w}^T\mathbf{w}}$$

We can simplify the expression:

$$\sup_{\left\{\mathbf{x}\in\mathbb{R}^n \middle| \|\mathbf{x}\|=1\right\}} R_A(\mathbf{x}) = \sup_{\{\mathbf{w}\in\mathbb{R}^n\}} \frac{\mathbf{w}^T\mathbf{A}\mathbf{w}}{\mathbf{w}^T\mathbf{w}}$$

$$\sup_{\left\{\mathbf{x}\in\mathbb{R}^n \middle| \|\mathbf{x}\|=1\right\}} R_A(\mathbf{x}) = \sup_{\{\mathbf{w}\in\mathbb{R}^n\}} R_A(\mathbf{w})$$

and recover our original optimisation problem by letting $\mathbf{x} = \mathbf{w}$, showing that it is equivalent to the supremum over the unit sphere. Assuming the set containing the unit sphere is compact, the extreme value theory of calculus states that $\sup_{\{\mathbf{x}\in\mathbb{R}^n \mid \|\mathbf{x}\|=1\}} R_A(\mathbf{x})$ is attained so equivalently $\sup_{\{\mathbf{x}\in\mathbb{R}^n\}} R_A(\mathbf{x})$ is attained as required.

(b) We can now reformulate the optimisation as:

$$\sup_{\left\{\mathbf{x}\in\mathbb{R}^n \middle| \|\mathbf{x}\|=1\right\}} \frac{\mathbf{x}^T\mathbf{A}\mathbf{x}}{\|\mathbf{x}\|^2}$$

Because $\|\mathbf{x}\| = 1$ (i.e. choosing $\mathbf{w} \in \mathbb{R}^n$ and $\mathbf{x} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$ to reformulate the problem over the unit sphere), we can equivalently write:

$$\sup_{\left\{\mathbf{x}\in\mathbb{R}^n \middle| \|\mathbf{x}\|=1\right\}} \mathbf{x}^T\mathbf{A}\mathbf{x}$$

Thus, showing $R_A(\mathbf{x}) \leq \lambda_1$ will be equivalent to showing $\mathbf{x}^T\mathbf{A}\mathbf{x} \leq \lambda_1$ for $\|\mathbf{x}\| = 1$. We know that for all $\mathbf{x} \in \mathbb{R}^n$:

$$\mathbf{x} = \sum_{i=1}^{n} (\xi_i^T \mathbf{x})\xi_i$$

so we can write:

$$\mathbf{x}^T A x = \Big(\sum_{i=1}^{n} (\xi_i^T \mathbf{x})\xi_i^T\Big)\mathbf{A}\Big(\sum_{i=1}^{n} (\xi_i^T \mathbf{x})\xi_i\Big)$$

Given that ξ_iR are eigenvectors of $\mathbf{A}$ corresponding to eigenvalues λ_i:

$$\mathbf{x}^T\mathbf{A}\mathbf{x} = \Big(\sum_{i=1}^{n} (\xi_i^T \mathbf{x})\xi_i^T\Big)\Big(\sum_{i=1}^{n} \lambda_i(\xi_i^T \mathbf{x})\xi_i\Big)$$

Given that the eigenvectors ξ_i form an orthonormal basis, we know that $\xi_i^T\xi_j = 0$ when $i \neq j$ and $\xi_i^T\xi_j = 1$ when $i = j$, so:

$$\mathbf{x}^T\mathbf{A}\mathbf{x} = \sum_{i=1}^{n} \lambda_i(\xi_i^T \mathbf{x})^2$$

From our above reformulation with the unit sphere, we know that $\|\mathbf{x}\|^2 = 1$ so $\|\mathbf{x}\|^2 = \sum_{i=1}^{n} x_i^2 = \sum_{j=i}^{n}(\mathbf{x})^2 = 1$. Thus the quantity $\sum_{i=1}^{n} \lambda_i(\xi_i^T\mathbf{x})^2$ is a weighted average of λ_i's with weights $(\xi_i^T\mathbf{x})^2$, which is always less than or equal to the largest λ_i value so:

$$\mathbf{x}^T\mathbf{A}\mathbf{x} = \sum_{i=1}^{n} \lambda_i(\xi_i^T \mathbf{x})^2 \leq \lambda_1$$

where λ_1 is the largest eigenvalue of eigenvalues λ_i. Therefore, $R_A(\mathbf{x}) \leq \lambda_1$ as required.

(c) Given that $\mathbf{x} \in span\{\xi_{k+1}, ..., \xi_n\}$, we can rewrite $\mathbf{x}$:

$$\mathbf{x} = \sum_{i=k+1}^{n} (\xi_i^T \mathbf{x})\xi_i$$

Using the same line of argument as in part (b) we can bound $\mathbf{x}^T\mathbf{A}\mathbf{x}$:

$$\mathbf{x}^T\mathbf{A}\mathbf{x} = \sum_{i=k+1}^{n} \lambda_i(\xi_i^T \mathbf{x})^2 \leq \max\{\lambda_{k+1}, ..., \lambda_n\}$$

But given that the maximum eigenvalue λ_1 is not contained in $\{\lambda_{k+1}, ..., \lambda_n\}$:

$$\max\{\lambda_{k+1}, ..., \lambda_n\} < \lambda_1$$

and therefore $R_A(\mathbf{x}) < \lambda_1$ as required.

Appendix 1: constants.py

```python
import os

DATA_FOLDER = "data"

BINARY_DIGITS_FILE_PATH = os.path.join(DATA_FOLDER, "binarydigits.txt")
MESSAGE_FILE_PATH = os.path.join(DATA_FOLDER, "message.txt")
SYMBOLS_FILE_PATH = os.path.join(DATA_FOLDER, "symbols.txt")
TRAINING_TEXT_FILE_PATH = os.path.join(DATA_FOLDER, "war_and_peace.txt")

OUTPUTS_FOLDER = "outputs"

DEFAULT_SEED = 0
```

src/constants.py

Appendix 2: main.py

```python
import os

import numpy as np

from src.constants import (
    BINARY_DIGITS_FILE_PATH,
    MESSAGE_FILE_PATH,
    OUTPUTS_FOLDER,
    SYMBOLS_FILE_PATH,
    TRAINING_TEXT_FILE_PATH,
)
from src.solutions import q1, q2, q3, q5

if __name__ == "__main__":
    if not os.path.exists(OUTPUTS_FOLDER):
        os.makedirs(OUTPUTS_FOLDER)
    x = np.loadtxt(BINARY_DIGITS_FILE_PATH)

    # Question 1
    Q1_OUTPUT_FOLDER = os.path.join(OUTPUTS_FOLDER, "q1")
    if not os.path.exists(Q1_OUTPUT_FOLDER):
        os.makedirs(Q1_OUTPUT_FOLDER)
    q1.d(
        x,
        figure_path=os.path.join(Q1_OUTPUT_FOLDER, "q1d.png"),
        figure_title="Q1d: Maximum Likelihood Estimate",
    )
    q1.e(
        x,
        alpha=3,
        beta=3,
        figure_path=os.path.join(Q1_OUTPUT_FOLDER, "q1e"),
        figure_title="Q1e: Maximum A Prior",
    )

    # Question 2
    Q2_OUTPUT_FOLDER = os.path.join(OUTPUTS_FOLDER, "q2")
    if not os.path.exists(Q2_OUTPUT_FOLDER):
        os.makedirs(Q2_OUTPUT_FOLDER)
    q2.c(x, table_path=os.path.join(Q2_OUTPUT_FOLDER, "q2c.csv"))

    # Question 3
    Q3_OUTPUT_FOLDER = os.path.join(OUTPUTS_FOLDER, "q3")
    if not os.path.exists(Q3_OUTPUT_FOLDER):
        os.makedirs(Q3_OUTPUT_FOLDER)
    q3.e(
        x,
        alpha_parameter=1 + 1e-5,
        beta_parameter=1 + 1e-5,
        number_of_trials=4,
        ks=[2, 3, 4, 7, 10],
        epsilon=1e-5,
        max_number_of_steps=int(1e2),
        figure_path=os.path.join(Q3_OUTPUT_FOLDER, "q3e"),
        figure_title="Q3e",
        compression_csv_path=os.path.join(Q3_OUTPUT_FOLDER, "q3e-compression"),
    )

    # Question 5
    Q5_OUTPUT_FOLDER = os.path.join(OUTPUTS_FOLDER, "q5")
    if not os.path.exists(Q5_OUTPUT_FOLDER):
        os.makedirs(Q5_OUTPUT_FOLDER)
    with open(TRAINING_TEXT_FILE_PATH) as fp:
        training_text = fp.read().replace("\n", "").lower()
    with open(SYMBOLS_FILE_PATH) as fp:
        symbols = fp.read().split("\n")
    with open(MESSAGE_FILE_PATH) as fp:
        encrypted_message = fp.read()
    q5.a(
        symbols,
        training_text,
        transition_matrix_path=os.path.join(Q5_OUTPUT_FOLDER, "q5a-transition.csv"),
        invariant_distribution_path=os.path.join(Q5_OUTPUT_FOLDER, "q5a-invariant.csv"),
    )
    q5.d(
        encrypted_message,
        symbols,
        training_text,
        number_trials=10,
        number_of_mh_loops=int(1e4),
        number_start_attempts=int(1e4),
        log_decryption_interval=100,
        log_decryption_size=60,
        trial_decryptions_table_path=os.path.join(Q5_OUTPUT_FOLDER, "q5d-trials.csv"),
        decryptor_table_path=os.path.join(Q5_OUTPUT_FOLDER, "q5d-decrypter.csv"),
        decrypted_message_iterations_table_path=os.path.join(
            Q5_OUTPUT_FOLDER, "q5d-iterations.csv"
        ),
    )
```

main.py