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. To match our data sample space, we might have to define an additional mapping between our data and model spaces, adding unnecessary complexity. 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, one for each image:

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

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 of this, 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 and solve for $\hat{p}_d$:

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

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

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

Moreover, the second derivative with respect to p_d:

$$\frac{\partial \mathcal{L}(\{\mathbf{x}^{(n)}\}_{n=1}^N|\mathbf{p})}{\partial p_d^2} = \frac{-\sum_{n=1}^N x_d^{(n)}}{p_d^2} + \frac{-\sum_{n=1}^N (1-x_d^{(n)})}{(1-p_d)^2}$$

For a maximum, we need to show that the second derivative is negative. Since $x_d^{(n)} \in \{0,1\}$, in the worst case, of $N=1$, the single pixel $x_d^{(1)}$ must either be white ($\sum_{n=1}^N x_d^{(n)} > 0$) or black ($\sum_{n=1}^N 1 - x_d^{(n)} > 0$) with the other being zero, $\frac{\partial \mathcal{L}(\{\mathbf{x}^{(n)}\}_{n=1}^N|\mathbf{p})}{\partial p_d^2} < 0$ will be guaranteed and $\hat{p}_d$ is a maximum as required for the maximum likelihood estimate.

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:

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

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:

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

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

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

To show that this is a maximum, the second derivative is:

$$\frac{\partial^2 \mathcal{L}(\mathbf{p}|\{\mathbf{x}^{(n)}\}_{n=1}^{N})}{(\partial p_d)^2} = \frac{(1-\alpha) - \sum_{n=1}^{N} x_d^{(n)}}{(p_d)^2} + \frac{(1-\beta) - \sum_{n=1}^{N}(1 - x_d^{(n)})}{(1-p_d)^2}$$

. For a maximum, we need $\frac{\partial^2 \mathcal{L}(\mathbf{p}|\{\mathbf{x}^{(n)}\}_{n=1}^{N})}{(\partial p_d)^2} < 0$ meaning that we need at least one of the strict inequalities $\alpha < 1 - \sum_{n=1}^{N} x_d^{(n)}$ or $\beta < 1 - \sum_{n=1}^{N}(1 - x_d^{(n)})$ to be satisified, where the other can be $\leq$. The Beta distribution requires $\alpha > 0$ and $\beta > 0$ so this requirement will always be satisfied (in the worst case of a single image, either $x_d^{(1)} = 1$ or $1 - x_d^{(1)} = 1$).

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:
    """
    X: numpy array of shape (N, D)
    """
    return np.mean(x, axis=0)


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

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


def d(x, figure_path, figure_title):
    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, alpha, beta, figure_path, figure_title):
    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:

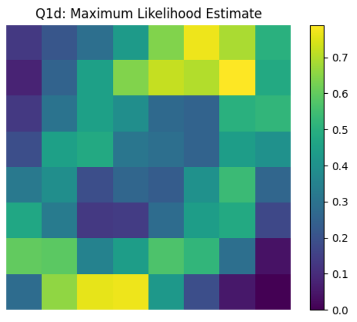

Figure 1: ML parameters

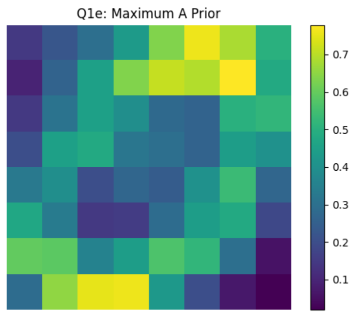

Figure 2: MAP parameters

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 $\frac{1}{N}\sum_{n=1}^{N}\mathbf{x}^{(n)}$, $\hat{\mathbf{p}}^{MLE}$. This makes sense because as our data set gets bigger, we are less reliant on our prior. 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.

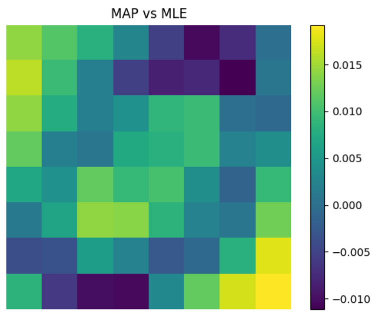

Figure 3: $\hat{\mathbf{p}}^{MAP} - \hat{\mathbf{p}}^{MLE}$

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-likelihoods 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.

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:

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

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

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

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

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

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

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

where $d = 1, ..., D$ and $\{\mathbf{x}^{(n)}\}_{n=1}^N$ is our data set.

Given that the prior of any unknown probabilities is uniform, i.e. $P(p_d|M_i) = 1$. We can simplify:

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

For M_1, we have that all pixels have probability 0.5:

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

We can remove the integrals and 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:

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

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

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:

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

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:

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

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 $i \in \{1,2,3\}$:

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

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

Table 1: Posterior Probabilities

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:

We can see that for models specified to have the same parameter value for all pixels like M_1 is very unlikely with the given data set. This makes sense because it is specifying models where the image is essentially blank (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 a subset of the models 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):
    n, d = x.shape
    return n * d * np.log(0.5)


def _log_p_d_given_m2(x):
    n, d = x.shape
    k = np.sum(x, axis=0).astype(int)
    return betaln(np.sum(k) + 1, n * d - np.sum(k) + 1)


def _log_p_d_given_m3(x):
    n, _ = x.shape
    k = np.sum(x, axis=0).astype(int)
    return logsumexp(betaln(k + 1, n - k + 1))


def c(x, table_path):
    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_d = log_p_d_given_m - logsumexp(log_p_d_given_m)
    df = pd.DataFrame(
        data=np.array(
            [
                np.arange(len(log_p_m_given_d)).astype(int) + 1,
                [f"1E{int(x/np.log(10))}" for x in log_p_m_given_d[:-1]]
                + [
                    f"1-{'-'.join([f'(1E{int(x/np.log(10))})' for x in log_p_m_given_d[:-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:

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

where $\{\mathbf{x}^{(n)}\}_{n=1}^{N}$ is our data set with $\mathbf{x}^{(n)} \in \mathbb{R}^{D\times 1}$ and $\theta = \{\pi, \mathbf{P}\}$, $\pi = [\pi_1, ..., \pi_K] \in \mathbb{R}^{K\times 1}$ our 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 under 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 $P(\mathbf{x}^{(n)}|\theta)$:

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

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

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

which 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:

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

(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 $P(s^{(n)} = k|\mathbf{x}^{(n)}, \pi) = \pi_k$. Note that $P(s^{(n)} = k|\mathbf{x}^{(n)}, \pi) = P(s^{(n)} = k|\mathbf{x}^{(n)}, \pi, \mathbf{P})$ as $s^{(n)}$ isn't dependent on $\mathbf{P}$.

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

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

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,

$$\log \tilde{r}_{nk} = \log \pi_k + \sum_{d=1}^{D} \Big(\mathbf{x}_d^{(n)} \log(p_{kd}) + (1 - \mathbf{x}_d^{(n)}) \log(1 - \exp(\log(p_{kd}))) \Big)$$

and

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

which can be vectorised as $\log \mathbf{r}_n$ 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:

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

Let this quantity be E. Each term of E can be expressed:

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

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 $\hat{P}$.

For the k^{th} element of π:

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

The second derivative:

$$\frac{\partial E}{(\partial \pi_k)^2} = \sum_n r_{nk} \frac{-1}{(\pi_k)^2}$$

is always negative because $r_{nk} \geq 0$, $\sum_n r_{nk} = 1$, $\pi_k \geq 0$, and $\sum_n \pi_k = 1$, ensuring a maximum in the next step.

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 $\mathbf{P}$:

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

Simplifying:

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

Similar to Question 1, we can see that taking second derivative, the term in the brackets will always be less than zero and with $r_{nk} \geq 0$ and $\sum_n r_{nk} = 1$, the second derivative will always be negative. This ensures that we have a maximum in the next step.

Setting the derivative to zero:

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

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

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

Thus,

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

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 $\mathbf{P}$,$\hat{\mathbf{P}}_{dk}^{MAP}$ by solving:

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

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:

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

Substituting the appropriate expressions:

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

Simplifying:

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

For a maximum, we see that we need $\alpha > \mathbf{x}_d^{(n)} - 1$ or $\beta < \mathbf{x}_d^{(n)}$, both of which are satisfied knowing that $\alpha > 0$ and $\beta > 0$. Setting $\frac{\partial E'}{\partial \mathbf{P}_{dk}} = 0$ we can calculate $\hat{\mathbf{P}}_{dk}^{MAP}$:

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

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

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.

(d) Plotting the posterior likelihood as a function of the iteration number:

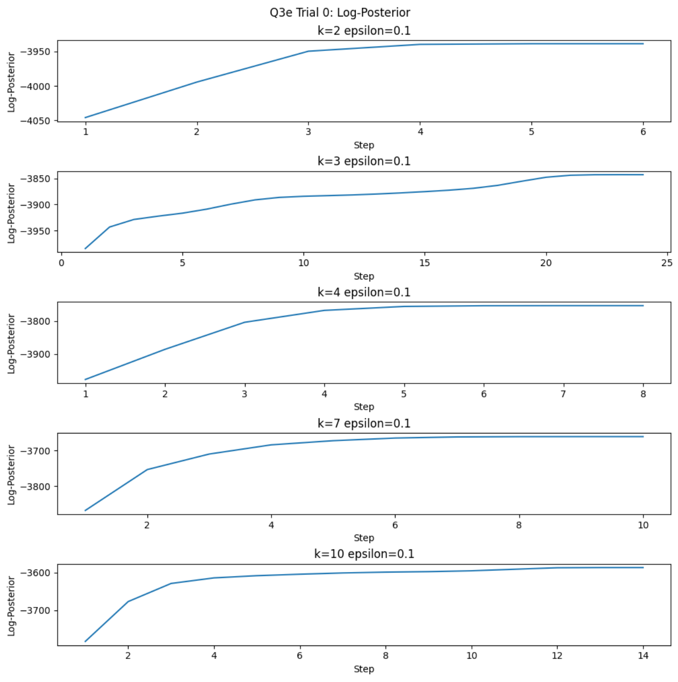

Figure 4: Log Likelihood vs Iteration Number

where *epsilon* is the stopping condition for the posterior posterior converges.

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

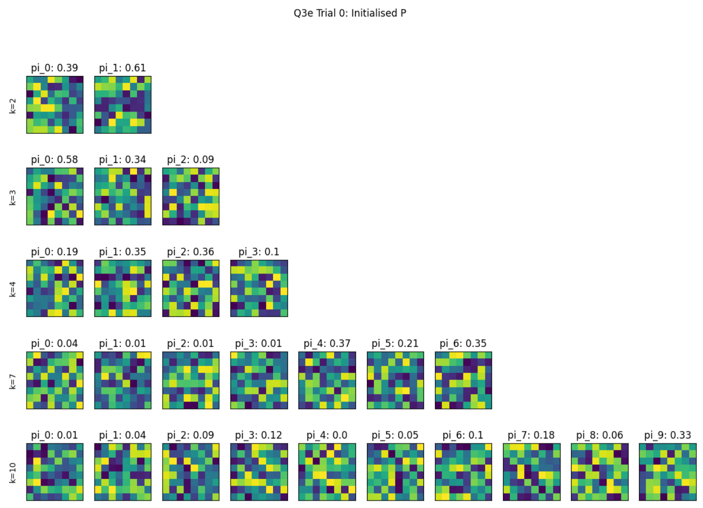

Figure 5: Randomly initialised parameters

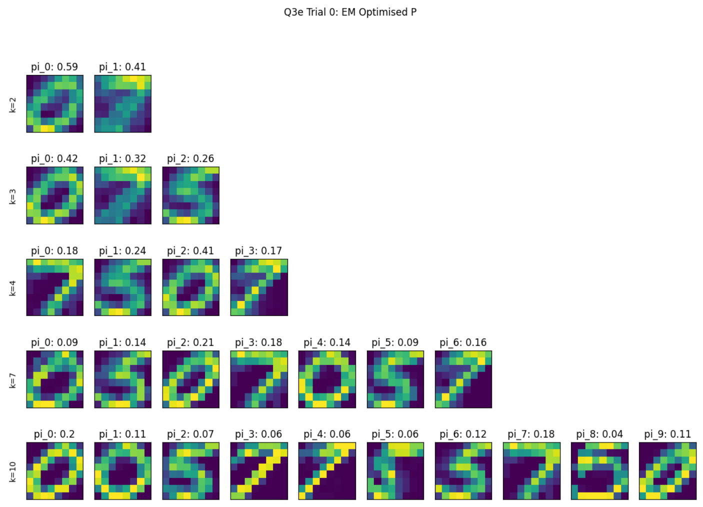

Figure 6: EM optimised parameters

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
from scipy.special import logsumexp
from sklearn.manifold import TSNE

from src.constants import DEFAULT_SEED


@dataclass
class Theta:
    """
    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):
        return np.exp(self.log_pi)

    @property
    def p_matrix(self):
        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):
        """
        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)
    """
    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_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
) -> 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)
    :return: an array of the maximised pixel probabilities for each component (d, k)
    """
    n, d = x.shape
    _, 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)

    alpha = 2
    beta = 2

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

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

    log_p_matrix_normalised_posterior = (
        log_p_matrix_unnormalised_posterior - log_p_matrix_normaliser_posterior
    )
    return log_p_matrix_normalised_posterior


def _compute_log_m_step(x: np.ndarray, log_responsibility: np.ndarray) -> 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)
    :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),
    )


def _run_expectation_maximisation(
    x: np.ndarray, theta: Theta, 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 max_number_of_steps: the maximum number of steps to run the algorithm
    :param epsilon: the minimum required change in log likelihood, otherwise the algorithm stops early
    :return: a tuple containing the optimised thetas, the log responsibilities,
            and the log likelihood at each step of the algorithm
    """
    log_responsibility = None
    log_likelihoods = []
    for _ in range(max_number_of_steps):
        log_responsibility = _compute_log_e_step(x, theta)
        theta = _compute_log_m_step(x, log_responsibility)

        log_likelihoods.append(_compute_log_likelihood(x, theta))

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


def _plot_p_matrix(
    thetas: List[Theta], ks: List[int], figure_title: str, figure_path: str
):
    n = len(ks)
    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 _plot_tsne_responsibility_clusters(
    log_responsibilities: List[np.ndarray],
    ks: List[int],
    figure_title: str,
    figure_path: str,
):
    n = len(ks)
    fig = plt.figure()
    fig.set_figwidth(5 * n)
    fig.set_figheight(5)
    for i, k in enumerate(ks):
        if k > 2:
            embedding = TSNE(
                n_components=2,
                learning_rate="auto",
                init="random",
                perplexity=10,
                random_state=DEFAULT_SEED,
            ).fit_transform(log_responsibilities[i])
        else:
            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:
    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"Log-Posterior")
        ax[i].set_title(f"{k=} {epsilon=}")
    plt.suptitle(figure_title)

    plt.savefig(figure_path)


def e(
    x: np.ndarray,
    number_of_trials: int,
    ks: List[int],
    epsilon: float,
    max_number_of_steps: int,
    figure_path: str,
    figure_title: str,
) -> None:
    n, d = x.shape
    np.random.seed(DEFAULT_SEED)
    for i in range(number_of_trials):
        init_thetas = []
        em_thetas = []
        log_posteriors = []
        log_responsibilities = []
        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,
                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)

        _plot_p_matrix(
            init_thetas,
            ks,
            figure_title=f"{figure_title} Trial {i}: Initialised P",
            figure_path=f"{figure_path}-{i}-initialised-p.png",
        )
        _plot_p_matrix(
            em_thetas,
            ks,
            figure_title=f"{figure_title} Trial {i}: EM Optimised P",
            figure_path=f"{figure_path}-{i}-optimised-p.png",
        )
        _plot_tsne_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}: Log-Posterior",
            figure_path=f"{figure_path}-{i}-log-pos.png",
        )
```
src/solutions/q3.py

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

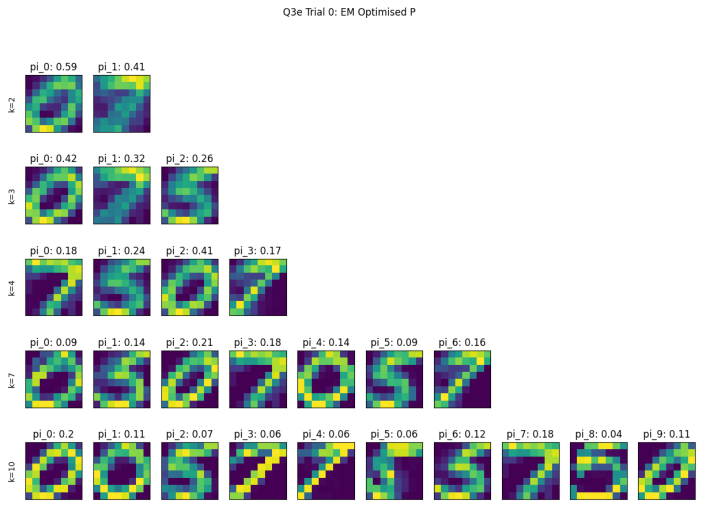

Figure 7: EM optimised parameters: Trial 0

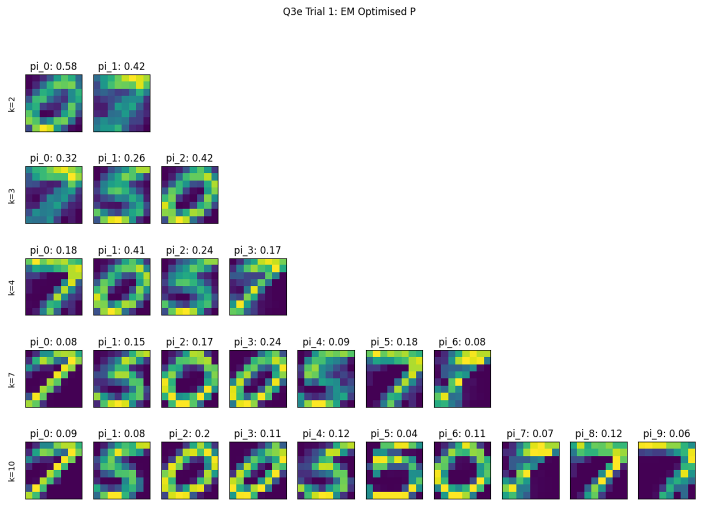

Figure 8: EM optimised parameters: Trial 1

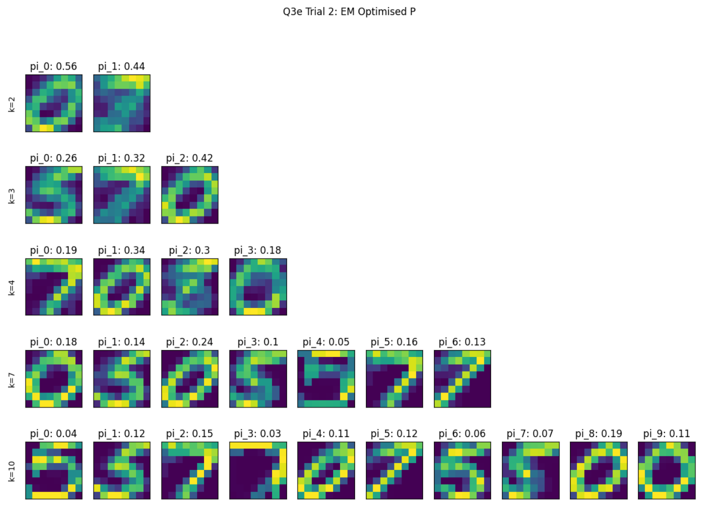

Figure 9: EM optimised parameters: Trial 2

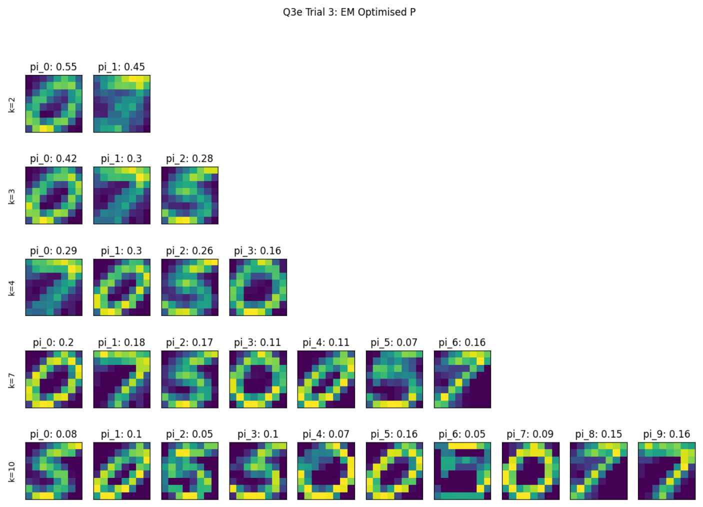

Figure 10: EM optimised parameters: Trial 3

For smaller k, we can visually see that we obtain very similar solutions (a 7 and a 0 for $k = 2$). However for higher K, we see that this may not always be the case. For Trial 2 of $k = 10$, we have three 5's whereas in Trial 4 we have two 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 or zero probability (i.e. π_2 of trial 2). It will be unlikely for those components to represent very distinct clusters (i.e. the parameters for π_2 and π_9 are very similar in trial 2) This can be verified when we perform a TSNE visualisation of the responsibility vector for each of the images (Note that for $k = 2$, the responsibility vector is displayed). We can see that for large k, qualitatively the number of clusters no longer matches the k value, indicating that some clusters are redundant. For example for $k = 7$ and $k = 10$ we can only qualitatively see four or five clusters with TSNE.

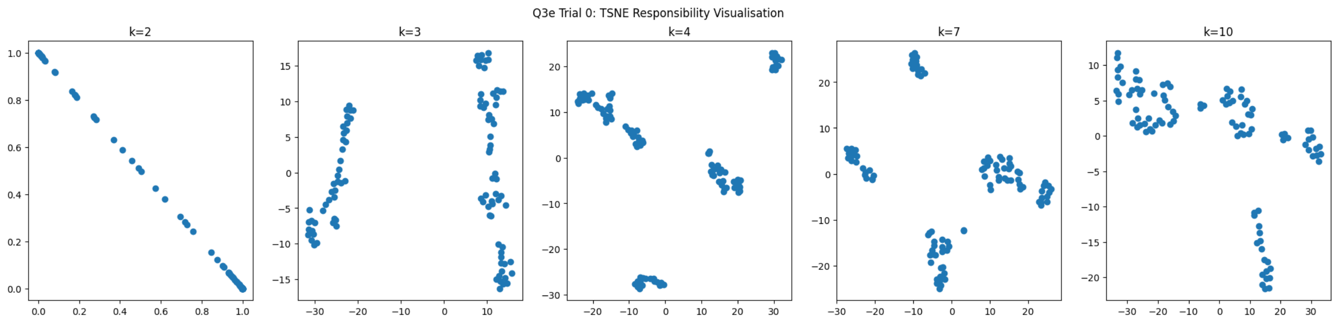

Figure 11: TSNE Visualisation of Image responsibilities: Trial 0

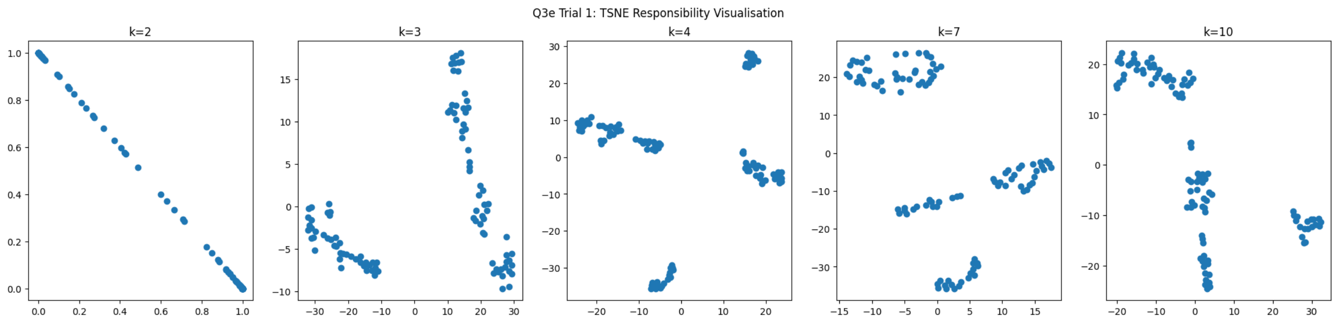

Figure 12: TSNE Visualisation of Image responsibilities: Trial 1

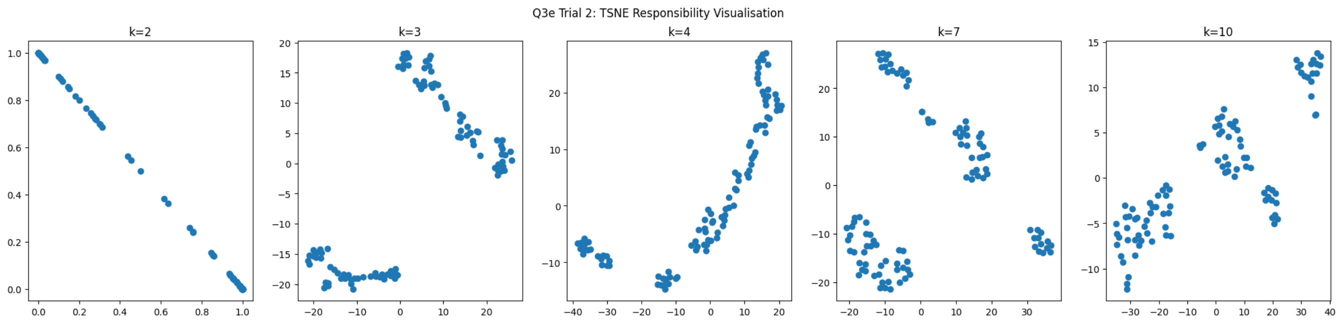

Figure 13: TSNE Visualisation of Image responsibilities: Trial 2

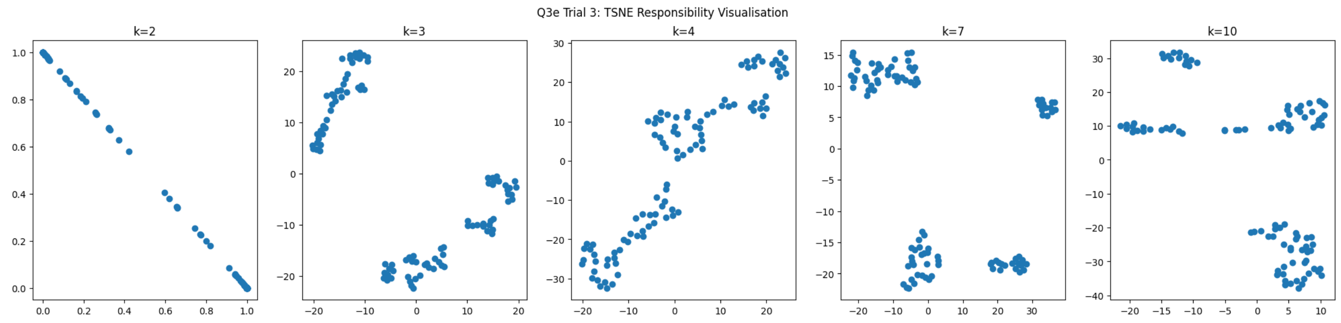

Figure 14: TSNE Visualisation of Image responsibilities: Trial 3

Improvements to the model could include searching for an optimal k by maximising the log posterior with regularisation on the magnitude of k to balancing maximising log posterior with minimising model complexity. Additionally, adding a prior on the responsibility components can be helpful to ensure non-zero mixing components unlike the components visualised here. This could help promote more meaningful clusters as k increases.

[BONUS] Express the log-likelihoods obtained in bits and relate these numbers to the length of the naive encoding of these binary data. How does your number compare to gzip (or another compression algorithm)? Why the difference? [5 marks]

[BONUS] Consider the total cost of encoding both the model parameters and the data given the model. How does this total cost compare to gzip (or similar)? How does it depend on K ? What might this tell you? [5 marks]

Question 5

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

$$\Psi(\alpha, \beta) = \frac{N_{s_i, s_{i-1}}}{N_{s_{i-1}}}$$

where $N_{s_i,s_{i-1}}$ is the count of the number of occurrences of the pair (s_i, s_{i-1}), where s_{i-1} is followed by s_i and $N_{s_{i-1}}$ is the number of occurrences of s_{i-1}.

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

(i) Initialise any $\phi_0 \in \mathbb{R}^{53\times1}$

(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, $\Psi_{i,j} = P(\alpha_j|\alpha_i)$ where α_i is the i^{th} symbol and α_j is the j^{th} symbol, 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 γ in the vector ϕ.

The transition matrix Ψ:

	=	space	-	”,”	;	:	!	?	/	.	’	double quotes	(	)	[	]	*	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
=	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	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	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	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
-	1.8e-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	1.2e-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
”,”	1.8e-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	1.2e-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
;	1.8e-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	1.2e-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
:	1.8e-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	1.2e-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
!	1.8e-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	1.2e-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
?	1.8e-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	1.2e-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
/	1.8e-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	1.2e-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
.	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	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
’	1.8e-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	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	1.8e-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	1.2e-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
(	1.8e-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	1.2e-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
)	1.8e-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	1.2e-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
[	1.8e-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	1.2e-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
]	1.8e-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	1.2e-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
*****	1.8e-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	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	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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	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	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	1.8e-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	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	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	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	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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	1.2e-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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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	1.8e-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	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 the symbols for a given σ.

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:

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

because $S(\sigma \to \sigma')$ 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 $\frac{1}{\binom{53}{2}}$, 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	d0[?0?,sdhrg0tdc0[,gr0as]drgti]r0?rtg'0[?0bt4org0htar0[r0',
100	odzhvzv,sdfrtz d5zh,trzasldrt ilrzvr twzhvzb egrtzf arzhrzw,
200	odgh.g.,sdfrtg dagh,trg/sldrt blrg.r tugh.gi e?rtgf /rghrgu,
300	odrh.r.-sdfgir darh-igrksldgi blgr.g iurh.rt eygirf kgrhgru-
400	idrkhrh-sdfgor dark-ogr.sldgo blgrhg ourkhrt eygorf .grkgru-
500	idrkhrh-sdflor dark-olrnsgdlo bglrhl ourkhrt eylorf nlrklru-
600	idrwhrhasdlofr d-rwaforpsgdof bgorho furwhrt eyofrl porworua
700	idrwhrhasdlotr d-rwatorpsgdot bgorho turwhrf eyotrl porworua
800	idrwhrhasd-otr dlrwatorpsgdot bgorho turwhrf eyotr- porworua
900	idrwhrhasdgotr dlrwatoruscdot bcorho tprwhrf eyotrg uorworpa
1000	idrwhrhasd.otr dlrwatorgscdot bcorho tprwhrf eyotr. gorworpa
1100	ilrwhrhasl.otr ldrwatorgsclot ncorho tfrwhrp eyotr. gorworfa
1200	ilrwhrhasl.otr ldrwatorgsclot ncorho tfrwhrp eyotr. gorworfa
1300	ilrwhrhasl.ofr ldrwaforgsclof ncorho ftrwhrp eyofr. gorworta
1400	ilrwhrhasl.ofr ldrwaforgsclof ncorho ftrwhrp eyofr. gorworta
1500	inrwhrhasngofr ndrwafor.scnof bcorho ftrwhrp eyofrg .orworta
1600	inrchrhasngofr ndrcafor.swnof bworho ftrchrp eyofrg .orcorta
1700	inrchrhasngofr ndrcafor.swnof bworho ftrchrp eyofrg .orcorta
1800	inrchrhasngofr ndrcaforlswnof bworho ftrchrp eyofrg lorcorta
1900	inrchrhasngofr ndrcaforlswnof bworho ftrchrp eyofrg lorcorta
2000	inrchrhasngofr ndrcaforlswnof bworho ftrchrp eyofrg lorcorta
2100	inechehasngofe ndecafoelswnof bwoeho ftechep ryofeg loecoeta
2200	in ch hasn.of end cafo lswnofebwo hoeft ch peryof .elo co ta
2300	in wh hasn.of end wafo lscnofevco hoeft wh geryof .elo wo ta
2400	in wh hasn.of end wafo lscnofevco hoeft wh geryof .elo wo ta
2500	in wh hasn.of end wafo lscnofevco hoeft wh geryof .elo wo ta
2600	in wh hasn.of end wafo lscnofevco hoeft wh geryof .elo wo ta
2700	in wh hasn.of end wafo lscnofevco hoeft wh geryof .elo wo ta
2800	in ch hasn.ol end calo fswnolevwo hoelt ch geryol .efo co ta
2900	in ch hasn.ol end calo fswnolevwo hoelt ch geryol .efo co ta
3000	in ch hasn.ol end calo fswnolevwo hoelt ch geryol .efo co ta
3100	in ch haun.ol end calo fuwnolevwo hoelt ch geryol .efo co ta
3200	in ch haun.ol end calo fuwnolevwo hoelt ch geryol .efo co ta
3300	in ch haun.os end caso fuwnosevwo hoest ch geryos .efo co ta
3400	in ch haun.os end caso fuwnosevwo hoest ch geryos .efo co ta
3500	in ch haun.os end caso fuwnosevwo hoest ch geryos .efo co ta
3600	in cy yaun.or end caro fuwnorevwo yoert cy geshor .efo co ta
3700	in cy yaun.er ond care luwnerovwe yeort cy gosher .ole ce ta
3800	in cy yaun.er ond care lubnerovbe yeort cy fosher .ole ce ta
3900	in cy yaun.er ond care lubnerovbe yeort cy fosher .ole ce ta
4000	in cy yaun.er ond care bufnerovfe yeort cy losher .obe ce ta
4100	in my yaun.er ond mare bufnerovfe yeort my losher .obe me ta
4200	in my yaun.er ond mare bufnerovfe yeort my losher .obe me ta
4300	in my yaun.er ond mare bufnerovfe yeort my losher .obe me ta
4400	in my yaun.er ond mare bufnerovfe yeort my losher .obe me ta
4500	in my yaun.er ond mare vufnerobfe yeort my losher .ove me ta
4600	in my yaun.er ond mare vufnerobfe yeort my losher .ove me ta
4700	in my yaun.er ond mare vufnerobfe yeort my losher .ove me ta
4800	in my yaunger ond mare vufnerobfe yeors my lother gove me sa
4900	in my yaunger ond mare vufnerobfe yeors my lother gove me sa
5000	in my yaunger ond mare vufnerobfe yeors my lother gove me sa
5100	in my yaunger ond mare vufnerobfe yeors my lother gove me sa
5200	in my yaunger ond mare vufnerobfe yeors my lother gove me sa
5300	in my yaunger ond mare vufnerobfe yeors my lother gove me sa
5400	in my yaunger ond mare vufnerobfe yeors my lother gove me sa
5500	in my yaunger ond mare vufnerobfe yeors my lother gove me sa
5600	in my yaunger ond mare vufnerobfe yeors my lother gove me sa
5700	in my yaunger ond mare vufnerobfe yeors my lother gove me sa
5800	in my yaunger ond mare vufnerobfe yeors my lother gove me sa
5900	in my yaunger ond mare vufnerobfe yeors my lother gove me sa
6000	in my yaunger ond mare vufnerobfe yeors my lother gove me sa
6100	in my yaunger ond mare vufnerobfe yeors my lother gove me sa
6200	in my yaunger ond mare vufnerobfe yeors my lother gove me sa
6300	in my yaunger ond mare vufnerobfe yeors my lother gove me sa
6400	in my yaunger ond mare vulneroble yeors my fother gove me sa
6500	in my yaunger ond mare vulneroble yeors my fother gove me sa
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
’	2
double quotes	double quotes
(	3
)	=
[	i
]	o
*	1
0	z
1	m
2	c
3	8
4	)
5	.
6	*
7	k
8	0
9	q
a	/
b	:
c	-
d	;
e	5
f	6
g	s
h	9
i	’
j	]
k	[
l	y
m	v
n	d
o	4
p	space
q	?
r	g
s	t
t	7
u	p
v	j
w	a
x	u
y	!
z	w

To help with chain initialisation, 10000 different σ’s were randomly and independently sampled. The σ providing the best log-likelihood was chosen as the starting point for the MH chain and algorithm was then run for 10000 iterations. Moreover, ten different trials were performed, where the trial with the best log-likelihood is displayed.

The Python code for the MH sampler:

```python
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


class Decrypter:
    def __init__(self, decryption_dict):
        self.decryption_dict = decryption_dict

    def decrypt(self, encrypted_message):
        return "".join([self.decryption_dict[x] for x in encrypted_message])


class Statistics:
    def __init__(
        self,
        training_text: str,
        symbols: List[str],
        invariant_stopping_epsilon: float = 5e-20,
    ):
        self.training_text = training_text
        self.symbols = symbols
        self.num_symbols = len(symbols)
        self.symbols_dict = {k: v for v, k in enumerate(symbols)}
        self.text_numbers = [
            self.symbols_dict[symbol]
            for symbol in list(training_text)
            if symbol in self.symbols_dict
        ]
        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)

    def _construct_transition_matrix(
        self, training_text: str, symbols_dict: Dict[str, int]
    ) -> np.ndarray:

        # initialise with ones to ensure ergodicity
        transition_matrix = np.ones((self.num_symbols, self.num_symbols))
        for i in range(1, len(training_text)):
            # check symbols are valid
            if (
                training_text[i] in symbols_dict
                and training_text[i - 1] in symbols_dict
            ):
                transition_matrix[
                    symbols_dict[training_text[i - 1]], symbols_dict[training_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:
        invariant_distribution = np.zeros((self.num_symbols, 1))
        previous_invariant_distribution = invariant_distribution.copy()
        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:
        return self.log_transition_matrix[
            self.symbols_dict[beta], self.symbols_dict[alpha]
        ]

    def log_invariant_probability(self, gamma: str) -> float:
        return self.log_invariant_distribution[self.symbols_dict[gamma]].item()

    def compute_log_probability(self, message: str) -> float:
        log_probability = self.log_invariant_probability(message[0])
        for i in range(1, len(message)):
            s_i = message[i]
            s_i_minus_1 = message[i - 1]
            log_probability += self.log_transition_probability(s_i, s_i_minus_1)
        return log_probability


class MetropolisHastingsDecryption:
    def __init__(self, symbols):
        self.symbols = symbols
        self._random_generator = np.random.default_rng()

    def generate_random_decrypter(self) -> Decrypter:
        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:
        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)

    def _choose_decrypter(
        self,
        statistics,
        encrypted_message,
        current_decrypter: Decrypter,
        proposal_decrypter: Decrypter,
    ) -> Decrypter:
        current_log_probability = statistics.compute_log_probability(
            message=current_decrypter.decrypt(encrypted_message),
        )
        proposal_log_probability = statistics.compute_log_probability(
            message=proposal_decrypter.decrypt(encrypted_message),
        )
        acceptance_probability = np.min(
            [1, np.exp(proposal_log_probability - current_log_probability)]
        )
        return self._random_generator.choice(
            [current_decrypter, proposal_decrypter],
            p=[1 - acceptance_probability, acceptance_probability],
        )

    def _find_good_starting_decrypter(
        self,
        statistics: Statistics,
        encrypted_message,
        number_start_attempts,
    ) -> Decrypter:
        best_log_likelihood = -np.float("inf")
        best_decrypter = None
        for _ in range(number_start_attempts):
            decrypter = self.generate_random_decrypter()
            if (
                statistics.compute_log_probability(
                    message=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,
        check_decryption_interval: int,
        check_decryption_size: int,
    ) -> Tuple[Decrypter, List[str]]:
        decrypter = self._find_good_starting_decrypter(
            statistics, encrypted_message, number_start_attempts
        )
        logged_decryption_message = [
            decrypter.decrypt(encrypted_message)[:check_decryption_size]
        ]
        for i in range(1, number_of_mh_loops + 1):
            if (i + 1) % check_decryption_interval == 0:
                logged_decryption_message.append(
                    decrypter.decrypt(encrypted_message)[:check_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 _convert_to_scientific_notation(x: float) -> str:
    return "{:.1e}".format(float(x))


```

```python
def a(
    symbols: List[str],
    training_text: str,
    transition_matrix_path: str,
    invariant_distribution_path: str,
):
    statistics = Statistics(
        training_text,
        symbols,
    )
    symbols_for_df = statistics.symbols.copy()
    symbols_for_df[symbols_for_df.index(" ")] = "space"
    symbols_for_df[symbols_for_df.index('"')] = "double quotes"
    df = pd.DataFrame(
        data=statistics.transition_matrix,
        columns=symbols_for_df,
    )
    df.index = symbols_for_df
    df.applymap(_convert_to_scientific_notation).to_csv(transition_matrix_path)

    df = (
        pd.DataFrame(
            data=statistics.invariant_distribution.reshape(1, -1),
            columns=symbols_for_df,
        )
        .applymap(_convert_to_scientific_notation)
        .transpose()
        .reset_index()
    )
    df.columns = ["Symbol", "Probability"]
    df.set_index("Symbol").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,
    check_decryption_interval: int,
    check_decryption_size: int,
    decryptor_table_path: str,
    decrypted_message_iterations_table_path: str,
):
    statistics = Statistics(
        training_text,
        symbols,
    )
    np.random.seed(DEFAULT_SEED)
    metropolis_hastings_decryption = MetropolisHastingsDecryption(symbols)
    decrypters = []
    log_likelihoods = []
    logged_decryption_messages = []
    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,
            check_decryption_interval,
            check_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)[:check_decryption_size]
        )

    # sort trials by log likelihood
    best_trial = np.argmax(log_likelihoods)

    decrpyter_table = pd.DataFrame(
        decrypters[best_trial].decryption_dict.items(), columns=["s", "sigma(s)"]
    )
    decrpyter_table[decrpyter_table == " "] = "space"
    decrpyter_table[decrpyter_table == '"'] = "double quotes"
    decrpyter_table.set_index("s").to_csv(decryptor_table_path, sep="|")

    decrypted_message_iterations_table = pd.DataFrame(
        [
            np.arange(0, len(logged_decryption_messages[best_trial]))
            * check_decryption_interval,
            logged_decryption_messages[best_trial],
        ]
    ).transpose()
    decrypted_message_iterations_table.columns = ["MH Iteration", "Current Decryption"]
    decrypted_message_iterations_table.set_index("MH Iteration").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 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) Analyse this approach to decoding. For instance, would symbol probabilities alone (rather than transitions) be sufficient? If we used a second order Markov chain for English text, what problems might we encounter? Will it work if the encryption scheme allows two symbols to be mapped to the same encrypted value? Would it work for Chinese with > 10000 symbols? [13 marks]

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 is much more difficult because each letter is assumed to be independent of its neighbours. For a first order Markov chairn, 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 decodings can't have the same likelihood), then using symbol probabilities alone should theoretically work. However, in practise we would unlikely to be able to make these assumptions about symbol frequencies from the size of our training set and encrypted message.

A second-order chain should also work in theory. However, this approach is probably practically more difficult for finding a suitable decoding. This is because our transition matrix would contain N^3, 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 second order transitions in a text of length N is $N - 2$ and third order its $N - 3$).

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 $\sigma^{-1}(e)$ is ill-defined, and therefore how we computing the joint probability of the encrypted text is no longer immediately clear. 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. Overall, many changes would need to be made to the approach to accommodate for these complications. It is not immediately obvious how current approach could work for this case.

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. An alternative set up could be with using Chinese phonetics, for which there are likely 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 $g(\mathbf{x}) = \mathbf{0}$ 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:

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

$$\frac{-x}{y} = 0$$

We see that $x = 0$. 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 our $\lambda = -1$ and $(x = 0, y = -1)$ when our $\lambda = 1$.

(b)

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

$$x = \ln(a)$$

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

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:

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

Simplifying:

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

Appendix: 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,
        number_of_trials=4,
        ks=[2, 3, 4, 7, 10],
        epsilon=1e-1,
        max_number_of_steps=int(1e2),
        figure_path=os.path.join(Q3_OUTPUT_FOLDER, "q3e"),
        figure_title="Q3e",
    )

    # 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),
        check_decryption_interval=100,
        check_decryption_size=60,
        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