University College London

Department of Computer Science

COMP0086: Probabilistic and Unsupervised Learning

Summative Assignment

Student Number:
18018699

Last Updated:
November 13, 2024

Contents

1 Models for Binary Vectors (25 marks)

Important things to restate about the problem

- Each image is a vector of pixels, concatenated by the columns.
- We have N images $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}$
- $\mathbf{x}^{(i)} \in \{0,1\}^D, \quad D = \text{rows} \times \text{cols. of image}$
- $\mathbf{x}^{(i)} = (x^{(1)}, \ldots, x^{(D)})$

1.1 Part A: Why MVG would not be an appropriate model?

A multivariate Gaussian (MVG) would not be the most appropriate model because one may argue that the data we are modelling is not continuous but binary. MVG also has a singular mode, and we have 2 states which our data can be, which also makes it inappropriate in that sense.

1.2 Part B: Maximum likelihood (ML) estimate of p

Assume each image is modelled i.i.d. from a D-dimensional Multivariate Bernoulli (MVB) distribution.

$$f(\mathbf{x} \mid \mathbf{p}) = \prod_{d=1}^{D} p_d^{x_d}(1-p_d)^{(1-x_d)}$$

To find the equation for the MLE of $\mathbf{p}$. We are solving for $\mathbf{p}$ directly.
Let's say S is our dataset of images.

$$S = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$$

$\hat{p}_{\text{MLE}}$ is found by solving,

$$\nabla_{\mathbf{p}} f(S \mid \mathbf{p}) = 0$$

In other words, solving the system of equations

$$\frac{\partial}{\partial p_d} f(S \mid \mathbf{p}) = 0$$

Making this specific to our case ...

$$\nabla_{\mathbf{p}} f(S \mid \mathbf{p}) = \nabla_{\mathbf{p}} \left(\prod_{i=1}^{N} f(\mathbf{x}^{(i)} \mid \mathbf{p}) \right)$$

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

This is the equation we must solve to find $\hat{p}_{\text{MLE}}$.
We can also instead solve $\nabla_{\mathbf{p}} \log(f) = 0$ since the logarithm is monotonic.

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

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

It is clear that under $\nabla_{\mathbf{p}}$ we will come down to D equations like the above with some summations.

Before it gets messy though, let's compute these partial derivatives.

If $j \neq d$:

$$\frac{\partial}{\partial p_j}(\dots) = 0$$

$j = d$:

$$\frac{\partial}{\partial p_d}(\dots) = \frac{x_d^{(i)}}{p_d} - \frac{(1 - x_d^{(i)})}{(1 - p_d)}$$

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

Our original problem above therefore becomes D equations of the form ...

$$\sum_{i=1}^{N} \left(\frac{x_d^{(i)}}{p_d} - \frac{(1 - x_d^{(i)})}{(1 - p_d)} \right) = 0$$

$$\Rightarrow \sum_{i=1}^{N} \frac{(1 - p_d) x_d^{(i)} - p_d (1 - x_d^{(i)})}{p_d (1 - p_d)}$$

$$= (p_d (1 - p_d))^{-1} \sum_{i=1}^{N} \Big(x_d^{(i)} - x_d^{(i)} p_d - p_d + x_d^{(i)} p_d \Big)$$

$$\Rightarrow (p_d (1 - p_d))^{-1} \sum_{i=1}^{N} \Big(x_d^{(i)} - p_d \Big) = 0$$

$$\therefore (p_d (1 - p_d))^{-1} \sum_{i=1}^{N} x_d^{(i)} = (p_d (1 - p_d))^{-1} N p_d$$

$$\therefore p_d = \frac{1}{N} \sum_{i=1}^{N} x_d^{(i)}$$

$$\therefore \hat{p}_{\mathrm{ML}} = (p_1, \dots, p_D)$$

1.3 Part C: Maximum a posteriori (MAP) estimate for p

Assuming independent Beta priors on p_d

$$f_d(p_d) = \mathcal{B}(\alpha, \beta)^{-1} p_d^{\alpha-1}(1-p_d)^{\beta-1}$$

where

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

We want to find $\hat{p}_{\text{MAP}}$, the maximum a posteriori estimate of $\mathbf{p}$.

$$\hat{p}_{\text{MAP}} = \arg\max_{\mathbf{p}} f(\mathbf{p} \mid S)$$

$$= \arg\max_{\mathbf{p}} f(\mathbf{p}) f(S \mid \mathbf{p})$$

In essence, we get the more concretely that $\hat{p}_{\text{MAP}}$ solves

$$\nabla_{\mathbf{p}} \left(\prod_{d=1}^{D} f_d(p_d) \cdot \left(\prod_{i=1}^{N} f(\mathbf{x}^{(i)} \mid \mathbf{p}) \right) \right) = 0$$

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

Instead of solving this equation directly, we can solve the logarithmic version since we only care about finding the stationary point.

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

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

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

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

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

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

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

Once again, we have essentially a series of D PDEs.

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

We can simplify these PDEs further to D equations and drop the sum over d as if we are not dealing with the situation that, $j \neq d$ then we will treat these like constants...

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

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

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

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

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

$$\therefore \hat{p}_{\text{MAP}} = (p_1, \ldots, p_D)$$

1.4 Part D: Learning the maximum likelihood parameters

To compute our ML parameters, we should think about vectorising our computations. So instead of using this definition...

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

We will use...

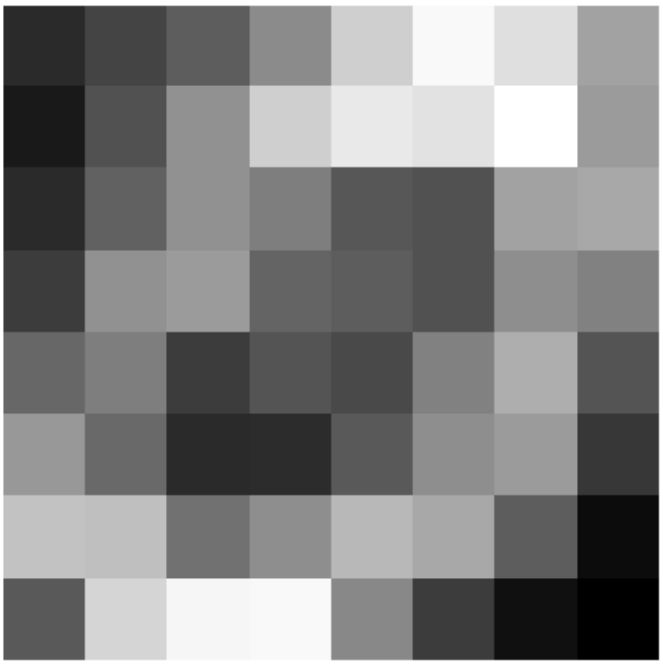
$$\hat{p}_{\text{ML}} = \frac{1}{N} X^\top \mathbf{1}_N.$$

Here is the Python3 code we will be running today to do this...

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

# load the data set
X = np.loadtxt('binarydigits.txt')

def p_mle(X):
    """
    Computes the MLE estimate vector for a matrix X.
    """
    N = X.shape[0]  # Number of samples (rows)
    return np.sum(X, axis=0) / N  # Equivalent to (1/N) * X.T @ ones(N)

p_mle_estimate = p_mle(X)

plt.figure()
plt.imshow(np.reshape(p_mle_estimate, (8,8)),
            interpolation="None",
            cmap='gray')
plt.axis('off')
```

This code produces the following image...

Figure 1: Image generated of the maximum likelihood estimate



1.5 Part E: Learning the maximum a posteriori parameters

To compute our MAP parameters, we should think about vectorising our computations. So instead of using this definition...

$$\hat{p}_{\text{MAP}} = \left(\frac{\alpha - 1 + \sum_{i=1}^{N} x_1^{(i)}}{(\alpha + \beta + N - 2)}, \ldots, \frac{\alpha - 1 + \sum_{i=1}^{N} x_D^{(i)}}{(\alpha + \beta + N - 2)} \right)$$

we will use...

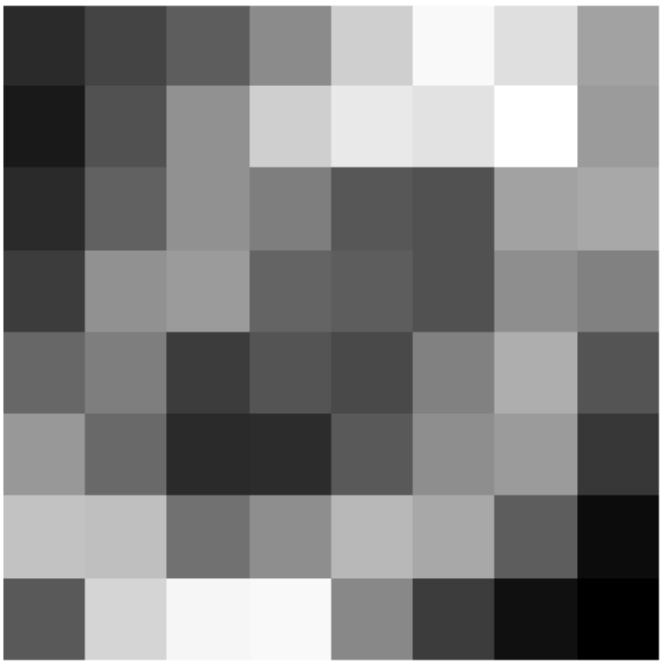
$$\hat{p}_{\text{MAP}} = \frac{\alpha - 1 + X^\top \mathbf{1}_N}{\alpha + \beta + N - 2}$$

Here is the Python3 code we will be running today to do this...

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

# load the data set
X = np.loadtxt('binarydigits.txt')

def p_map(X, alpha, beta):
    """Computes the maximum a posteriori estimate of the paramater vector for
    a matrix X. Where X is a matrix of binary digits, we assume all pixels
    are being sampled from a multivariate Bernoulli distribution.
    """
    N = X.shape[0]  # Number of samples (rows)

    numerator = alpha - 1 + np.sum(X, axis=0)
    denominator = alpha + beta + N - 2
    return numerator / denominator

p_map_estimate = p_map(X, alpha=3, beta=3)

# Plot the MAP estimate
plt.figure()
plt.imshow(
    np.reshape(p_map_estimate, (8,8)),
    interpolation="None",
    cmap='gray',
)
plt.axis('off')
plt.savefig('q1e_map_estimate.png')
```

This code produces the following image...

Figure 2: Image generated of the maximum a priori estimate



Why this might be better or worse than the ML estimate? This estimate may be better in the sense that we have included prior information, which, if correct, should model the binary digits more accurately. On the other hand, if this prior is wrong then we would be biasing the information of the data incorrectly, leading to a worse estimate.

2 Model Selection (15 marks)

In this section, we will compare and calculate the relative probability of the following three variations of our multivariate Bernoulli model from the first section of this summative assessment. The three variations are, where all components of the multivariate Bernoulli are generated by...

1. The same Bernoulli distribution with $p_d = 0.5$.
2. The same Bernoulli distribution with $p_d = k$ where $0 \leq k \leq 1$.
3. Individual Bernoulli distributions with separate and unknown p_d.

Under the following assumptions...

- Assume that all three models are equally likely a prior.
- Take the prior distributions for any unknown probabilities to be uniform.

Under all these conditions, we want to find the posterior probabilities of each of the three models having generated the data in 'binarydigits.txt'. This posterior can be written as...

$$p(\mathcal{M}_i \mid S) = \frac{p(S \mid \mathcal{M}_i) \cdot p(\mathcal{M}_i)}{p(S)} \iff \text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}$$

Where $S = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$, and $\mathcal{M}_i$ is the specific model we are evaluating.

Since we are only interested in the relative probabilities, we don't have to care about calculating the evidence. We are also graced with the fact that our priors on the models are equal, so to compare the relative posterior probabilities for all three models, we only have to compute the likelihoods. Due to numerical instability, we will first evaluate the relative log-posterior for all the models. We don't exponentiate those results to get the relative posterior probabilities because any time we do this due to a limit in my machine's precision we get 0. For briefness, we will refer to the relative posterior as the posterior in the later parts of this question.

$$\text{Relative Posterior} = \text{Likelihood} \times \text{Prior} \implies \log(\text{Relative Posterior}) = \log(\text{Likelihood}) + \log(\text{Prior})$$

We will also be using the following formula from the slides to evaluate our likelihood for the different models when our parameters θ_i of the model vary...

$$p(\mathcal{D} \mid \mathcal{M}_i) := \int d\theta_i \cdot p(\mathcal{D} \mid \theta_i, \mathcal{M}_i) \cdot p(\theta_i \mid \mathcal{M}_i)$$

2.1 Part A: Bernoulli distributions with $p_d = 0.5$

Mathematical Simplification:
Given our model, $\mathcal{M}_A$ we can say that the likelihood has the form...

$$p(S \mid \mathcal{M}_A) = \prod_{i=1}^{N} \prod_{d=1}^{D} p_d^{x_d^{(i)}} (1 - p_d)^{(1 - x_d^{(i)})}$$

For $p_d = \frac{1}{2}$...

$$p(S \mid \mathcal{M}_A) = \prod_{i=1}^{N} \prod_{d=1}^{D} \left(\frac{1}{2}\right)^{x_d^{(i)}} \left(\frac{1}{2}\right)^{(1 - x_d^{(i)})} = \prod_{i=1}^{N} \prod_{d=1}^{D} \left(\frac{1}{2}\right)^{x_d^{(i)}} \left(\frac{1}{2}\right) \left(\frac{1}{2}\right)^{-x_d^{(i)}}$$

$$\therefore p(S \mid \mathcal{M}_A) = \prod_{i=1}^{N} \prod_{d=1}^{D} \left(\frac{1}{2}\right) = \left(\frac{1}{2}\right)^{(N \times D)}$$

$$\therefore p(\mathcal{M}_A \mid S) = \left(\frac{1}{2}\right)^{(N \times D)} \times \left(\frac{1}{3}\right) \implies \log\left(p(\mathcal{M}_A \mid S)\right) = N \cdot D \cdot \log\left(\frac{1}{2}\right) + \log\left(\frac{1}{3}\right)$$

Code:

```python
import numpy as np

# load the data set
X = np.loadtxt('binarydigits.txt')

N, D = X.shape

# Likelihood when p_d = 0.5
log_likelihood = N*D*np.log(1/2)
log_prior = np.log(1/3)
log_posterior = log_likelihood + log_prior

print(f'Log-Posterior when p_d = 0.5: {log_posterior:.2f}')
```

Result:
Log-Posterior when $p_d = 0.5$: -4437.24 ($\implies$ $0 <$ Posterior $<< 1$)

2.2 Part B: Bernoulli distributions with $p_d = k$

Mathematical Simplification:

$$p(S \mid \mathcal{M}_B) = \int_0^1 dp_d \cdot p(S \mid p_d, \mathcal{M}_B) \cdot p(p_d \mid \mathcal{M}_B)$$

We know that $p(p_d \mid \mathcal{M}_B)$ is a uniform distribution over the interval $[0, 1]$, and so $p(p_d \mid \mathcal{M}_B) = 1$. Implying...

$$p(S \mid \mathcal{M}_B) = \int_0^1 dp_d \cdot p(S \mid p_d, \mathcal{M}_B) = \int_0^1 dp_d \cdot \prod_{i=1}^{N} \prod_{d=1}^{D} p_d^{x_d^{(i)}} (1 - p_d)^{(1 - x_d^{(i)})}$$

$$p(S \mid \mathcal{M}_B) = \int_0^1 dp_d \cdot p_d^{\sum_{i=1}^{N} \sum_{d=1}^{D} x_d^{(i)}} (1 - p_d)^{\sum_{i=1}^{N} \sum_{d=1}^{D} (1 - x_d^{(i)})}$$

We then notice that this is the same form as the Beta function...

$$B(m, n) = \int_0^1 t^{m-1}(1 - t)^{n-1} dt = \frac{\Gamma(m)\Gamma(n)}{\Gamma(m + n)}$$

$$\therefore p(S \mid \mathcal{M}_B) = B\left(1 + \sum_{i=1}^{N} \sum_{d=1}^{D} x_d^{(i)}, 1 + \sum_{i=1}^{N} \sum_{d=1}^{D} (1 - x_d^{(i)})\right)$$

The nature of our problem means we know that our inputs into the Beta function are integers, this simplifies the problem further (Since for integers n we have that $\Gamma(n) = (n - 1)!$) ...

$$\therefore p(S \mid \mathcal{M}_B) = \frac{\left(\sum_{i=1}^{N} \sum_{d=1}^{D} x_d^{(i)}\right)! \times \left(\sum_{i=1}^{N} \sum_{d=1}^{D} (1 - x_d^{(i)})\right)!}{\left(1 + \sum_{i=1}^{N} \sum_{d=1}^{D} x_d^{(i)} + \sum_{i=1}^{N} \sum_{d=1}^{D} (1 - x_d^{(i)})\right)!}$$

Where...

$$\sum_{i=1}^{N} \sum_{d=1}^{D} x_d^{(i)} = \text{Count of elements equal to 1 in } S := C_1$$

$$\sum_{i=1}^{N} \sum_{d=1}^{D} (1 - x_d^{(i)}) = \text{Count of elements equal to 0 in } S := C_0$$

Therefore, our log-likelihood is...

$$\log\left(p(S \mid \mathcal{M}_B)\right) = \log\left(C_1!\right) + \log\left(C_0!\right) - \log\left((1 + C_1 + C_0)!\right).$$

Due to hardware limitations though, for computing such large values for our factorials values, we will not use this solution. Instead, we result to using Python3's inbuilt "math" package which has a "lgamma" function which returns $\log\left(\Gamma(x)\right)$ for a given x. This means our numerical solution is...

$$\log\left(p(S \mid \mathcal{M}_B)\right) = \log\left(\Gamma(1 + C_1)\right) + \log\left(\Gamma(1 + C_0)\right) - \log\left(\Gamma(2 + C_1 + C_0)\right)$$

As before, we know we can then use this to find the log-posterior...

$$\log\left(p(\mathcal{M}_B \mid S)\right) = \log\left(p(S \mid \mathcal{M}_B)\right) + \log\left(\frac{1}{3}\right)$$

Code:

```python
import numpy as np
from math import lgamma

# load the data set
X = np.loadtxt('binarydigits.txt')

# Calculate the count of 1s and 0s in the data set
count_1 = np.count_nonzero(X == 1)
count_0 = np.count_nonzero(X == 0)

# Due to numerical instability, we calculate the log of the beta function
# using the log-gamma function...
# log(B(m, n)) = log(gamma(m)) + log(gamma(n)) - log(gamma(m + n))
log_beta = lambda m, n : lgamma(m) + lgamma(n) - lgamma(m + n)

log_likelihood = log_beta(count_1 + 1, count_0 + 1)
log_prior = np.log(1/3)
log_posterior = log_likelihood + log_prior

print(f'Log-Posterior when p_d = k: {log_posterior:.2f}')
```

Result:
Log-Posterior when $p_d = k$: -4284.82 ($\implies$ $0 <$ Posterior $<< 1$)

2.3 Part C: Individual Bernoulli distributions

Mathematical Simplification:

$$p(S \mid \mathcal{M}_C) = \int_0^1 d\mathbf{p} \cdot p(S \mid \mathbf{p}, \mathcal{M}_C) \cdot p(\mathbf{p} \mid \mathcal{M}_C)$$

Using the assumption that individual implies independence between the D component Bernoulli distributions of our MVB model means that...

$$p(S \mid p_d, \mathcal{M}_C) = \prod_{i=1}^{N} p_d^{x_d^{(i)}} (1 - p_d)^{(1 - x_d^{(i)})}$$

$$p(S \mid \mathcal{M}_C) = \int_0^1 \cdots \int_0^1 p(S \mid p_1, \mathcal{M}_C) \ldots p(S \mid p_D, \mathcal{M}_C) \cdot 1 \cdot dp_1 \ldots dp_D$$

$$p(S \mid \mathcal{M}_C) = \prod_{d=1}^{D} \int_0^1 \prod_{i=1}^{N} p_d^{x_d^{(i)}} (1 - p_d)^{(1 - x_d^{(i)})} dp_d$$

$$p(S \mid \mathcal{M}_C) = \prod_{d=1}^{D} \int_0^1 dp_d \cdot p_d^{\sum_{i=1}^{N} x_d^{(i)}} (1 - p_d)^{\sum_{i=1}^{N} (1 - x_d^{(i)})}$$

From part B we know that we can replace this with a Beta distribution...

$$p(S \mid \mathcal{M}_C) = \prod_{d=1}^{D} B\left(1 + \sum_{i=1}^{N} x_d^{(i)}, 1 + \sum_{i=1}^{N} (1 - x_d^{(i)})\right)$$

$$\log\left(p(S \mid \mathcal{M}_C)\right) = \sum_{d=1}^{D} \log\left(B\left(1 + \sum_{i=1}^{N} x_d^{(i)}, 1 + \sum_{i=1}^{N} (1 - x_d^{(i)})\right)\right)$$

This time in the code, we will use a "for" loop to iteratively sum all the $\log\left(B(\ldots)\right)$ terms.

Code:

```python
import numpy as np
from math import lgamma

# load the data set
X = np.loadtxt('binarydigits.txt')

N, D = X.shape

# log(B(m, n)) = log(gamma(m)) + log(gamma(n)) - log(gamma(m + n))
log_beta = lambda m, n : lgamma(m) + lgamma(n) - lgamma(m + n)

# Calculate all the log(Beta())
logB_sum = 0
for d in range(D):
    # Calculate the count of 1s and 0s in the data set
    count_1_d = np.count_nonzero(X[:, d] == 1)
    count_0_d = np.count_nonzero(X[:, d] == 0)

    logB_sum += log_beta(count_1_d + 1, count_0_d + 1)

log_likelihood = logB_sum
log_prior = np.log(1/3)
log_posterior = log_likelihood + log_prior

print(f'Log-Posterior with individual Bernoullis: {log_posterior:.2f}')
```

Result:
Log-Posterior with individual Bernoullis: -3852.29 ($\implies 0 < \text{Posterior} << 1$)

3 EM for Binary Data (65 marks + 5 bonus)

3.1 Part A: Likelihood for a mixture of K multivariate Bernoulli

The likelihood of a mixture model of K multivariate Bernoulli distributions, parameters $\pi = (\pi_1, \ldots, \pi_K)$ in a column vector, to denote the proportions of the different distributions on a point, where $0 \leq \pi_m \leq 1$ and the sum of all the elements of π are 1. Here the likelihood is over the N samples in our dataset $\mathbf{x}^{(i)}$.

$$p(\mathcal{D} \mid \mathbf{P}, \pi) = \prod_{i=1}^{N} P(\mathbf{x}^{(i)} \mid \pi, \mathbf{P})$$

$$p(\mathcal{D} \mid \mathbf{P}, \pi) = \prod_{i=1}^{N} \sum_{m=1}^{K} \pi_m \cdot \text{MultivariateBernoulli}(\mathbf{x}^{(i)} \mid \mathbf{P})$$

$$p(\mathcal{D} \mid \mathbf{P}, \pi) = \prod_{i=1}^{N} \sum_{m=1}^{K} \pi_m \cdot \prod_{d=1}^{D} p_{kd}^{x_d^{(i)}} (1 - p_{kd})^{(1 - x_d^{(i)})}$$

$$p(\mathcal{D} \mid \mathbf{P}, \pi) = \prod_{i=1}^{N} \sum_{m=1}^{K} \pi_m \cdot \prod_{d=1}^{D} p_{kd}^{x_d^{(i)}} (1 - p_{kd})^{(1 - x_d^{(i)})}$$

where

$$\mathbf{P} = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1D} \\ p_{21} & p_{22} & \cdots & p_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ p_{K1} & p_{K2} & \cdots & p_{KD} \end{bmatrix}$$

3.2 Part B: Responsibility of our mixture components on our data

The responsibility of mixture component k for a data vector $\mathbf{x}^{(n)}$ is given by...

$$r_{nk} = P(s^{(n)} = k \mid \mathbf{x}^{(n)}, \pi, \mathbf{P}) = \frac{P_k(\mathbf{x}^{(n)} \mid \mathbf{P}) \cdot \pi_k}{\sum_{m=1}^{K} P_m(\mathbf{x}^{(n)} \mid \mathbf{P}) \cdot \pi_m}$$

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

3.3 Part C: Maximizing parameters for the expected log-joint

Our goal here is to find the π and $\mathbf{P}$ to solve the following equation...

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

Here we are saying that our $q(\{s^{(n)}\})$ is the probability distribution of a data point n being of a component of the set $\{1, \ldots, K\}$, in other words this is the responsibility $\therefore r_{nk} = q(s^{(n)} = k)$. To start, it may be easier to try to simplify the expression we are maximising by looking at the definition of an expected value $\langle f(x) \rangle_{p(x)} = \sum_{i=1}^{N} f(x_i) P(X = x_i)$, giving us...

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

$$= \sum_{k=1}^{K} \sum_{n=1}^{N} \log P(\mathbf{x}^{(n)}, s^{(n)} = k \mid \pi, \mathbf{P}) \cdot q(s^{(n)} = k)$$

$$= \sum_{k=1}^{K} \sum_{n=1}^{N} \log \left(P(\mathbf{x}^{(n)} \mid s^{(n)} = k, \pi, \mathbf{P}) \cdot P(s^{(n)} = k \mid \pi, \mathbf{P}) \right) \cdot q(s^{(n)} = k)$$

$$= \sum_{k=1}^{K} \sum_{n=1}^{N} \log \left(\left(\prod_{d=1}^{D} p_{kd}^{x_d^{(n)}} (1 - p_{kd})^{(1 - x_d^{(n)})} \right) \cdot P(s^{(i)} = k \mid \pi, \mathbf{P}) \right) \cdot q(s^{(i)} = k)$$

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

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

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

Using this simpler form for the problem, we go back to the beginning of the question and can state that our goal now is to find π and $\mathbf{P}$ to solve the following equation...

$$\arg\max_{\pi,\mathbf{P}} \left(\sum_{k=1}^{K} \sum_{n=1}^{N} \left(\sum_{d=1}^{D} \left(x_d^{(n)} \log (p_{kd}) + (1 - x_d^{(n)}) \log (1 - p_{kd}) \right) + \log \pi_k \right) \cdot r_{nk} \right)$$

which is equivalent to finding the π and $\mathbf{P}$ to solve both...

$$\nabla_{\mathbf{P}} \left(\sum_{k=1}^{K} \sum_{n=1}^{N} \left(\sum_{d=1}^{D} \left(x_d^{(n)} \log\left(p_{kd}\right) + (1 - x_d^{(n)}) \log\left(1 - p_{kd}\right) \right) + \log \pi_k \right) \cdot r_{nk} \right) = 0$$

$$\nabla_{\pi} \left(\sum_{k=1}^{K} \sum_{n=1}^{N} \left(\sum_{d=1}^{D} \left(x_d^{(n)} \log\left(p_{kd}\right) + (1 - x_d^{(n)}) \log\left(1 - p_{kd}\right) \right) + \log \pi_k \right) \cdot r_{nk} \right) = 0$$

This could also be thought of as solving $K \times D$ equations of the form...

$$\frac{\partial}{\partial p_{ij}} \left(\sum_{k=1}^{K} \sum_{n=1}^{N} \left(\sum_{d=1}^{D} \left(x_d^{(n)} \log\left(p_{kd}\right) + (1 - x_d^{(n)}) \log\left(1 - p_{kd}\right) \right) + \log \pi_k \right) \cdot r_{nk} \right) = 0$$

and K equations of the form...

$$\frac{\partial}{\partial \pi_i} \left(\sum_{k=1}^{K} \sum_{n=1}^{N} \left(\sum_{d=1}^{D} \left(x_d^{(n)} \log\left(p_{kd}\right) + (1 - x_d^{(n)}) \log\left(1 - p_{kd}\right) \right) + \log \pi_k \right) \cdot r_{nk} \right) = 0$$

In a normal mathematical setting, we would most likely have to solve this via the product rule. In our case where we are trying to obtain an iterative update for the parameters π and $\mathbf{P}$, the M-step of EM, we can treat the responsibility as a constant $\therefore r_{nk} \neq r_{nk}(\pi, \mathbf{P})$. This makes these derivatives much simpler.

First, we solve the PDE for a given p_{ij}...

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

$$= \frac{\partial}{\partial p_{ij}} \left(\sum_{n=1}^{N} \left(x_j^{(n)} \log\left(p_{ij}\right) + (1 - x_j^{(n)}) \log\left(1 - p_{ij}\right) \right) \cdot r_{ni} \right)$$

$$= \sum_{n=1}^{N} \left(\frac{x_j^{(n)}}{(p_{ij})} - \frac{(1 - x_j^{(n)})}{(1 - p_{ij})} \right) \cdot r_{ni}$$

From question 1 part B we know that we can write...

$$\sum_{i=1}^{N} \left(\frac{x_d^{(i)}}{p_d} - \frac{(1 - x_d^{(i)})}{(1 - p_d)} \right) = (p_d(1 - p_d))^{-1} \sum_{i=1}^{N} \left(x_d^{(i)} - p_d \right).$$

Applying this formula, we get that...

$$\sum_{n=1}^{N}\left(\frac{x_j^{(n)}}{(p_{ij})}-\frac{(1-x_j^{(n)})}{(1-p_{ij})}\right)\cdot r_{ni}=(p_{ij}(1-p_{ij}))^{-1}\sum_{n=1}^{N}\left(x_j^{(n)}-p_{ij}\right)\cdot r_{ni}.$$

$$\therefore (p_{ij}(1-p_{ij}))^{-1}\sum_{n=1}^{N}\left(x_j^{(n)}-p_{ij}\right)\cdot r_{ni}=0$$

$$\therefore \sum_{n=1}^{N}x_j^{(n)}\cdot r_{ni}=\sum_{n=1}^{N}p_{ij}\cdot r_{ni}$$

$$\therefore p_{ij}=\frac{\sum_{n=1}^{N}x_j^{(n)}\cdot r_{ni}}{\sum_{n=1}^{N}r_{ni}}$$

Let us solve the PDE for a given π_i, remembering to satisfy the summation condition across the mixing proportions π_k, by solving with the Lagrange multiplier all $K+1$ equations (remember $\lambda\neq 0$) ...

$$\frac{\partial}{\partial\pi_i}\left(\sum_{k=1}^{K}\sum_{n=1}^{N}\left(\sum_{d=1}^{D}\left(x_d^{(n)}\log\left(p_{kd}\right)+(1-x_d^{(n)})\log\left(1-p_{kd}\right)\right)+\log\pi_k\right)\cdot r_{nk}\right)-\lambda\cdot\frac{\partial}{\partial\pi_i}\left(\sum_{k=1}^{K}\pi_k-1\right)=0$$

$$\sum_{k=1}^{K}\pi_k-1=0$$

Differentiating we get...

$$\sum_{n=1}^{N}\frac{1}{\pi_i}\cdot r_{ni}-\lambda=0\implies\pi_i=\sum_{n=1}^{N}\frac{r_{ni}}{\lambda}$$

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

$$\therefore\pi_i=\sum_{n=1}^{N}\frac{r_{ni}}{N}$$

Because of the definition that $\sum_{k=1}^{K}r_{nk}=1$, where we have K components.

3.4 Part D: Implementing the EM Algorithm

First, to perform the EM Algorithm, we must know how to compute the log-likelihood. Given the likelihood from part A, we know that...

$$p(\mathcal{D} \mid \mathbf{P}, \pi) = \prod_{i=1}^{N} \sum_{m=1}^{K} \pi_m \cdot \prod_{d=1}^{D} p_{kd}^{x_d^{(i)}} (1 - p_{kd})^{(1 - x_d^{(i)})}$$

$$\therefore \log p(\mathcal{D} \mid \mathbf{P}, \pi) = \log \left(\prod_{i=1}^{N} \sum_{m=1}^{K} \pi_m \cdot \prod_{d=1}^{D} p_{kd}^{x_d^{(i)}} (1 - p_{kd})^{(1 - x_d^{(i)})} \right)$$

$$= \sum_{i=1}^{N} \log \left(\sum_{m=1}^{K} \pi_m \cdot \prod_{d=1}^{D} p_{kd}^{x_d^{(i)}} (1 - p_{kd})^{(1 - x_d^{(i)})} \right)$$

Now let us do the implementation, in Python3, of the EM algorithm as follows...

Helper Functions:

```python
import numpy as np
from matplotlib import pyplot as plt
from pprint import pprint

# load the data set
X = np.loadtxt('binarydigits.txt')

np.random.seed(1)

def E_step(X, pi, P):
    """Perform the E-step of the EM algorithm for a Multivariate Bernoulli Mixture Model.

    Input:
    X: N x D matrix of binary data
    pi: K x 1 vector of mixing coefficients
    P: K x D matrix of Bernoulli parameters

    Output:
    r: N x K matrix of responsibilities
    """
    N, D = X.shape
    K = len(pi)
    r = np.zeros((N, K))
    for n in range(N):
        denominator = np.sum([pi[j] * np.prod((P[j, :] ** X[n, :]) * ((1 - P[j, :]) ** (1 - X[n, :]))) for j in range(K)])
        for k in range(K):
            numerator = pi[k] * np.prod((P[k, :] ** X[n, :]) * ((1 - P[k, :]) ** (1 - X[n, :])))
            r[n, k] = numerator / denominator
    return r


def M_step(X, r):
    """Perform the M-step of the EM algorithm for a Multivariate Bernoulli Mixture Model.

    Input:
    X: N x D matrix of binary data
    r: N x K matrix of responsibilities

    Output:
    pi: K x 1 vector of mixing coefficients
    P: K x D matrix of Bernoulli parameters
```

```
    """
    N, D = X.shape
    K = r.shape[1]
    pi = np.sum(r, axis=0) / N
    P = np.zeros((K, D))
    for k in range(K):
        for d in range(D):
            P[k, d] = np.sum(X[:, d] * r[:, k]) / np.sum(r[:, k])
    return pi, P


def log_likelihood(X, pi, P):
    """Calculate the log-likelihood of the data under the current model.

    Input:
    X: N x D matrix of binary data
    pi: K x 1 vector of mixing coefficients
    P: K x D matrix of Bernoulli parameters

    Output:
    log_likelihood: log-likelihood of the data
    """
    N, D = X.shape
    K = len(pi)
    log_likelihood = 0

    # Calculate the log-likelihood
    for n in range(N):
        likelihood_for_1_image = 0
        for k in range(K):
            likelihood_for_1_image += pi[k] * np.prod((P[k, :]**X[n,
            ↪  :])*((1-P[k, :])**(1-X[n, :])))
        log_likelihood += np.log(likelihood_for_1_image)
    return log_likelihood
```

EM Algorithm:

```python
def em(X, K, epsilon=1e-16, max_iter=1000):
    """We implement the EM algorithm for a Multivariate Bernoulli Mixture
    Model. We are trying to find the pi, and P matrix (K x D) that
    maximizes  the log-likelihood of the data X, under some
    distribution q of the mixture model components.

    Input:
    X: N x D matrix of binary data
    K: number of components
    epsilon: minimum change in log-likelihood to continue the iterations
    max_iter: maximum number of iterations

    Output:
    pi: K x 1 vector of mixing coefficients
    P: K x D matrix of Bernoulli parameters
    log_likelihoods: list of log-likelihoods at each iteration
    """
    # Initialize the parameters
    N, D = X.shape
    log_likelihoods = []
    delta = np.inf
    i = 0

    # pi and P randomly sampled from a uniform distribution U(0, 1)
    # and appropriately normalized
    pi, P = np.random.rand(K), np.random.rand(K, D)
    pi = pi / pi.sum()

    while delta > epsilon and i < max_iter:
        # Calculate the responsibilities
        r = E_step(X, pi, P)

        # Calculate the parameters
        pi, P = M_step(X, r)

        # Calculate the log-likelihood
        log_likelihoods.append(log_likelihood(X, pi, P))

        # Check for convergence
        if i > 0: delta = np.abs(log_likelihoods[i] - log_likelihoods[i - 1])
        i += 1
    return pi, P, log_likelihoods
```

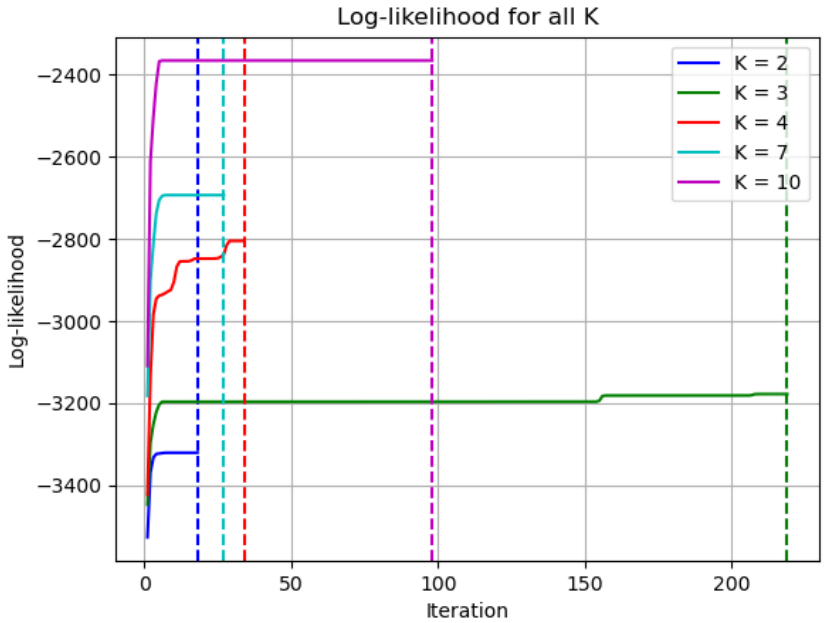
Running The EM Algorithm:

```python
K = [2, 3, 4, 7, 10]
K_log_likelihoods = []
for i in range(len(K)):
    # Run the EM algorithm
    pi, P, log_likelihoods = em(X, K[i])
    K_log_likelihoods.append(log_likelihoods)

    # Print the results
    print(f'K = {K[i]}, Iterations = {len(log_likelihoods)},')
    print(f'Final-Log-likelihood = {log_likelihoods[-1]}, Pi =')
    pprint(pi)
    print()

    # Plot the Bernoulli parameters in one figure
    fig, axs = plt.subplots(1, K[i], figsize=(10*K[i], int(5*np.cbrt(K[i]))))
    for j in range(len(P)):
        axs[j].imshow(P[j].reshape(8, 8), cmap='gray', interpolation='none')
        axs[j].set_title(f'Component {j+1}')
        axs[j].axis('off')
    plt.savefig(f'q3/bernoulli_parameters_K_{K[i]}.png')

# Plot the log-likelihood for all K
plt.figure()
plt.title('Log-likelihood for all K')
plt.grid()
colors = ['b', 'g', 'r', 'c', 'm']
for i in range(len(K)):
    l = len(K_log_likelihoods[i])
    plt.plot(range(1, l+1), K_log_likelihoods[i], label=f'K = {K[i]}',
    ↪  color=colors[i])
    # Plot the Iteration value where the log-likelihood converges
    plt.axvline(x=l, linestyle='--', color=colors[i])
plt.xlabel('Iteration')
plt.ylabel('Log-likelihood')
plt.legend()
plt.savefig('q3/log_likelihood_all_K.png')
```

Results from Code:

```
K = 2, Iterations = 18,
Final-Log-likelihood = -3320.3775169778814, Pi =
array([0.65962235, 0.34037765])

K = 3, Iterations = 219,
Final-Log-likelihood = -3177.5571427467785, Pi =
array([0.24989042, 0.15999792, 0.59011166])

K = 4, Iterations = 34,
Final-Log-likelihood = -2804.826323213882, Pi =
array([0.17      , 0.25000012, 0.39999988, 0.18      ])

K = 7, Iterations = 27,
Final-Log-likelihood = -2693.2531832001287, Pi =
array([0.05999988, 0.29000169, 0.02      , 0.21930991, 0.04      ,
       0.14068857, 0.22999995])

K = 10, Iterations = 98,
Final-Log-likelihood = -2365.828343399904, Pi =
array([0.04      , 0.16      , 0.09177852, 0.15074369, 0.03      ,
       0.01      , 0.11      , 0.06      , 0.16925631, 0.17822149])
```

Figure 3: All the different values of K and how their log-likelihoods change over the number of iterations. The vertical dashed lines represent where convergence stopped for the corresponding colour-matched model log-likelihood.



3.5 Part E: Comments on part D and plot the P matrices

First, here are our plots for the learned Bernoulli parameters for the mixture of multivariate Bernoulli models for different values of K...

Figure 4: $K = 2$

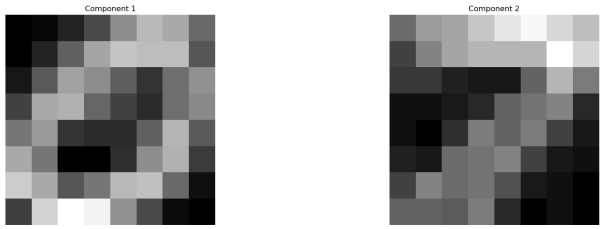


Figure 5: $K = 3$

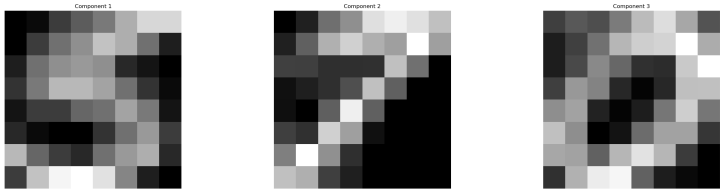


Figure 6: $K = 4$

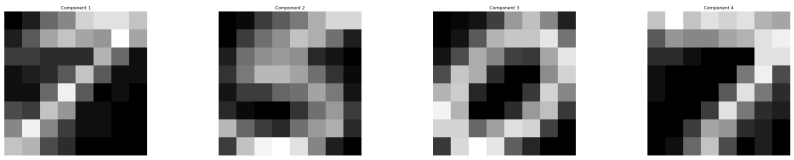


Figure 7: $K = 7$

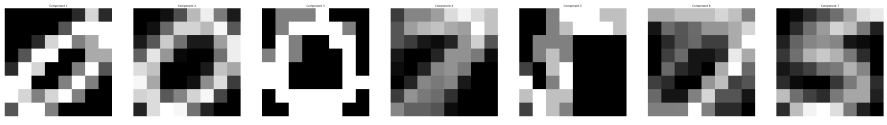
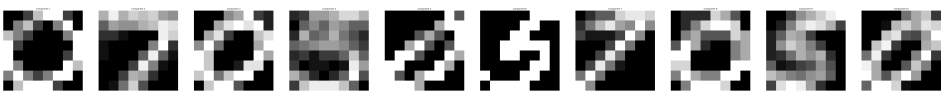


Figure 8: $K = 10$



Comments on our implementation:
Running this algorithm from various random seeds, it is clear that for different initial conditions, the solutions obtained show variation and a clear dependence on the value of K. For smaller K values (such as $K = 2$ or $K = 3$), the solutions tend to be more consistent, typically isolating distinct digits, 0 and 7 for $K = 2$, and 0, 5, and 7 for $K = 3$. However, with larger K values the solutions become less stable, and frequently showing multiple versions of the same handwritten digits, with many components π having very low probability values, this might point to some redundancy in the model having larger K values.

Enhancements could include implementing regularization to find more optimal K values, likely in the range $2 < K < 7$. Incorporating prior knowledge about digit variations, expanding the dataset to include all ten digits, and adding constraints to reduce redundancy for higher K values are also other ways we might go about improving the algorithm. The algorithm's sensitivity to initial conditions, especially at higher K values, also suggests that exploring improved initialization strategies might help achieve more consistent results.

4 LGSSMs, EM and SSID (35 bonus)

4.1 Part A: Make plots using the code provided

In this section, we are going to use the provided code in ”ssm_kalman.py” to plot the data given using a ”filt” Kalman filter and also a ”smooth” Kalman filter.

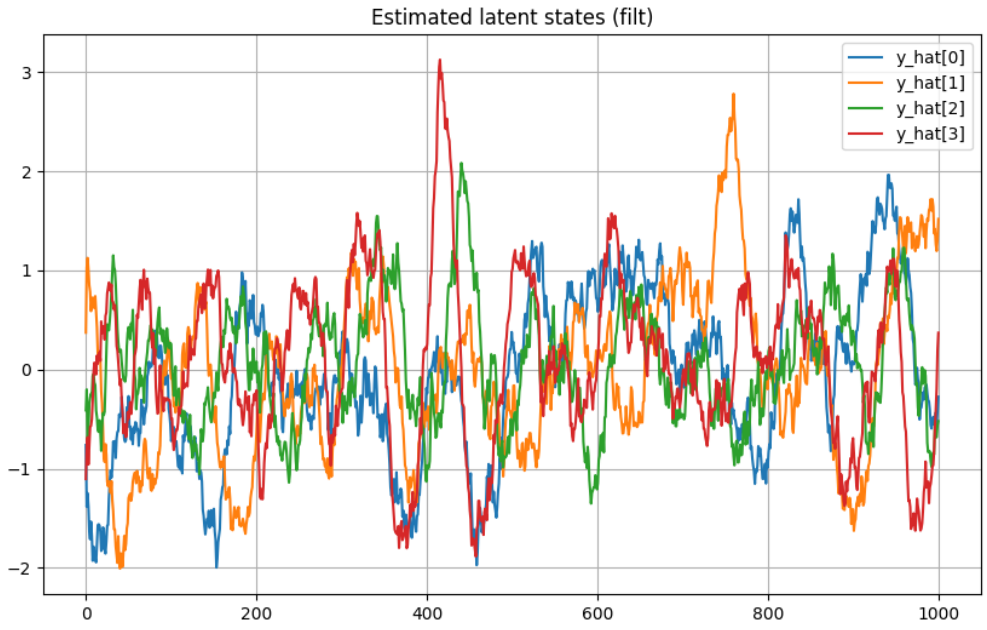
Code:

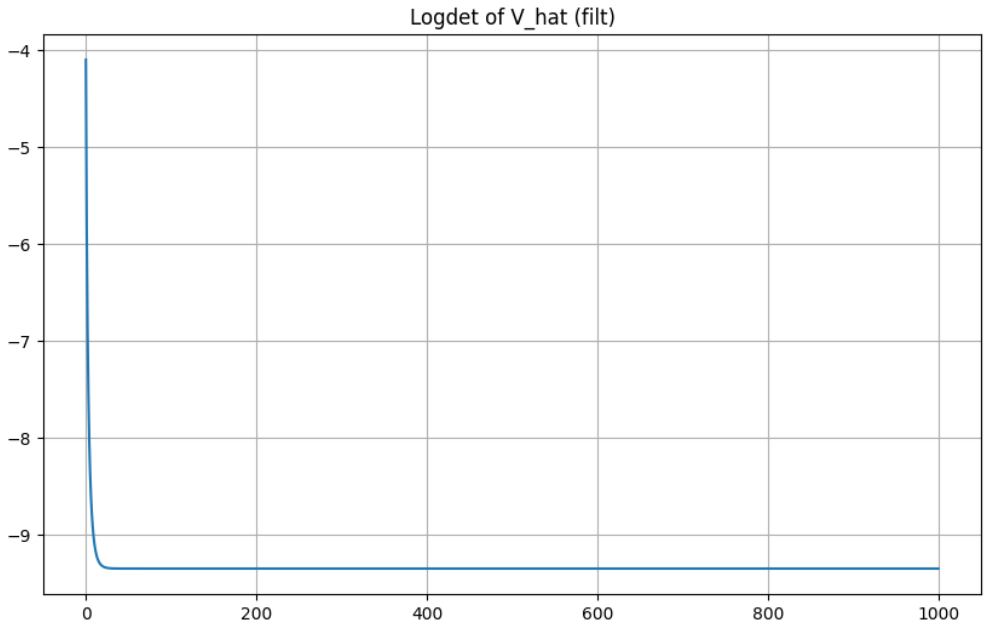
```
np.random.seed(0)

ssm_train_path = 'q4_ssm/ssm_spins.txt'
ssm_train = []
with open(ssm_train_path, 'r') as f:
    ssm_train = f.readlines()
    ssm_train = np.array([list(map(float, x.strip().split())) for x in
    ↪  ssm_train])

# SSM parameters and variables predefined in the problem sheet
y_init = np.random.randn(4)

# shorthand for the matrix A
s_180 = np.sin(2*np.pi/180)
s_90 = np.sin(2*np.pi/90)
c_180 = np.cos(2*np.pi/180)
c_90 = np.cos(2*np.pi/90)

A = 0.99 * np.array([
    [c_180, -s_180, 0, 0],
    [s_180, c_180, 0, 0],
    [0, 0, c_90, -s_90],
    [0, 0, s_90, c_90]
])

Q = np.eye(4) - A.dot(A.T)

R = np.eye(5)

C = np.array([
    [1, 0, 1, 0],
    [0, 1, 0, 1],
    [1, 0, 0, 1],
    [0, 0, 1, 1],
    [0.5, 0.5, 0.5, 0.5]
])
```

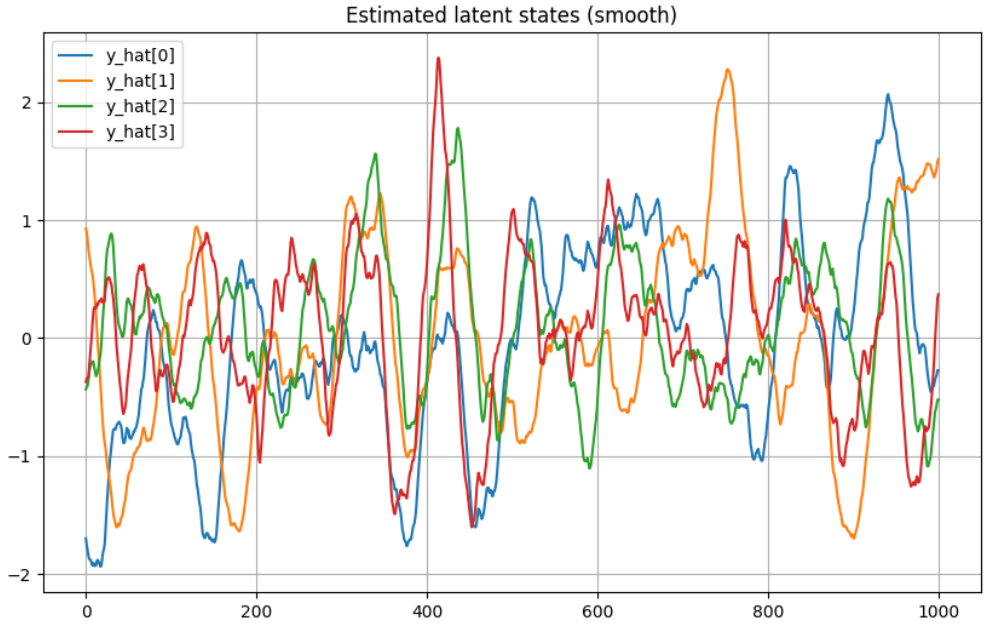
```
36
37 logdet = lambda A: 2 * np.sum(np.log(np.diag(np.linalg.cholesky(A))))
38 [y_hat, V_hat, V_joint, likelihood] = run_ssm_kalman(
39     X=ssm_train.T,
40     y_init=y_init,
41     Q_init=np.eye(4),
42     A=A,
43     Q=Q,
44     C=C,
45     R=R,
46     mode='filt'
47 )
48
49 # Plot the results for filt
50 plt.figure(figsize=(10, 6))
51 plt.plot(y_hat[0], label='y_hat[0]')
52 plt.plot(y_hat[1], label='y_hat[1]')
53 plt.plot(y_hat[2], label='y_hat[2]')
54 plt.plot(y_hat[3], label='y_hat[3]')
55 plt.legend()
56 plt.grid()
57 plt.title('Estimated latent states (filt)')
58 plt.savefig('q4_ssm/estimated_latent_states_filt.png')
59
60 # Plot the logdet of v_hat
61 plt.figure(figsize=(10, 6))
62 plt.plot([logdet(V_hat[i]) for i in range(V_hat.shape[0])])
63 plt.grid()
64 plt.title('Logdet of V_hat (filt)')
65 plt.savefig('q4_ssm/logdet_v_hat_filt.png')
66
67 # Smooth version
68 [y_hat, V_hat, V_joint, likelihood] = run_ssm_kalman(
69     X=ssm_train.T,
70     y_init=y_init,
71     Q_init=np.eye(4),
72     A=A,
73     Q=Q,
74     C=C,
75     R=R,
76     mode='smooth'
77 )
78
79 # Plot the results for smooth
80 plt.figure(figsize=(10, 6))
81 plt.plot(y_hat[0], label='y_hat[0]')
82 plt.plot(y_hat[1], label='y_hat[1]')
```

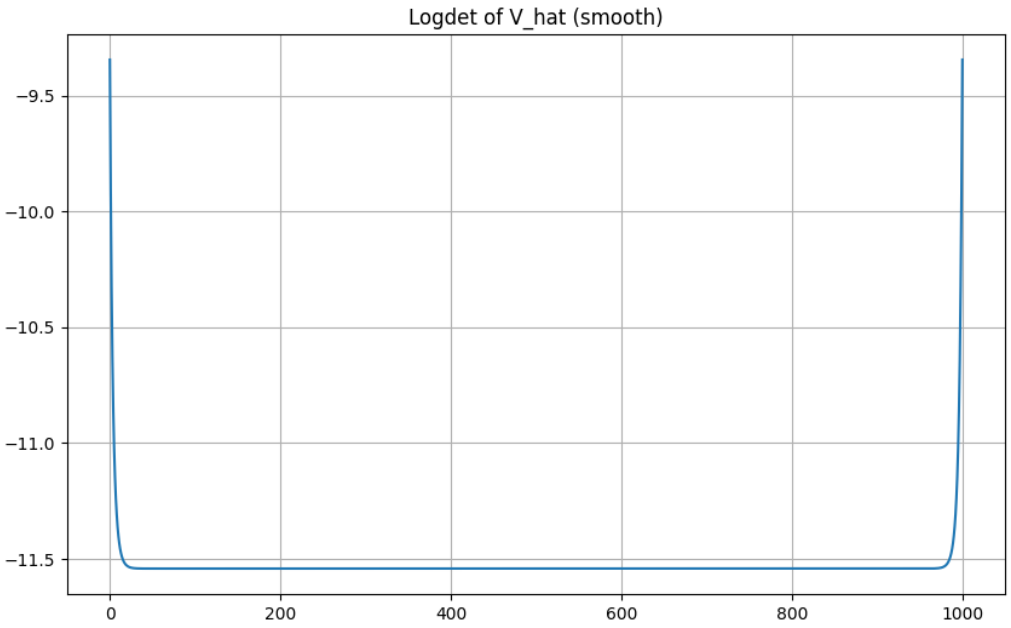
```
plt.plot(y_hat[2], label='y_hat[2]')
plt.plot(y_hat[3], label='y_hat[3]')
plt.legend()
plt.grid()
plt.title('Estimated latent states (smooth)')
plt.savefig('q4_ssm/estimated_latent_states_smooth.png')

# Plot the logdet of v_hat
plt.figure(figsize=(10, 6))
plt.plot([logdet(V_hat[i]) for i in range(V_hat.shape[0])])
plt.grid()
plt.title('Logdet of V_hat (smooth)')
plt.savefig('q4_ssm/logdet_v_hat_smooth.png')
```

Plots:




Logdet of V_hat (filt)
−4
−5
−6
−7
−8
−9
0
200
400
600
800
1000


Estimated latent states (smooth)
y_hat[0]
y_hat[1]
y_hat[2]
y_hat[3]
2
1
0
−1
−2
0
200
400
600
800
1000



Explanation of Behaviour:
These plots show the differences between Kalman filtering and smoothing in a Linear Gaussian State Space Model (LGSSM). The "Estimated latent states" plots show four state trajectories, where the smoothed version produces more stable estimates by using both past and future information, while the filtered version only uses past observations, resulting in more stochastic estimates. These differences are quantified in the "Logdet of V_hat" plots, where the filtered estimates stabilize around -9.3, while the smoothed estimates reach a lower value of -11.5, indicating higher confidence due to the additional information used. The smoothed version shows characteristic boundary spikes at t=0 and t=1000, which is expected behaviour due to reduced information availability at sequence endpoints. Overall, these plots show us how smoothing makes more precise state estimates at the cost of requiring the full observation sequence, while filtering offers immediate but slightly less accurate estimates using only past information.

5 Decrypting Messages with MCMC (70 marks)

In this section, we want to decrypt a message using MCMC. We follow the following assumptions and conditions...

- Mappings between symbols are $1 \to 1$, where $a \to b$, and $a \not\to (b, c)$.
- 'symbols.txt' has all of these symbols given per line. We have 53 symbols.
- The second symbol in 'symbols.txt' is $\langle \text{space} \rangle$.
- 'message.txt' is the encrypted message.
- Here, we will model our English text as symbols $S_1, S_2, ..., S_n$ under a Markov chain, $p(S_1, S_2, ..., S_n) = p(S_1) \prod_{i=2}^{n} p(S_i) \cdot p(S_i \mid S_{i-1})$.

5.1 Part A: Learn the ML parameters for the marginal and transition probabilities

To find the Maximum Likelihood estimates for these various transition and marginal probabilities, we must analyse the frequency of the individual letters and pairs of letters in a large corpus of text. In this case, we will use the recommended "War and Peace" book to find the ground truth of these English letters.

Where the marginal probabilities are learned with the following formula...

$$p(S_i = \alpha) = \frac{\text{Count}(\alpha)}{N},$$

where...

$$N = \text{Number of Symbols in "War and Peace"},$$

$$\text{Count}(s) = \text{Count the number of times } s \text{ appears in "War and Peace"}.$$

Correspondingly, the transition probabilities are learned with the following formula...

$$p(S_i = \alpha \mid S_{i-1} = \beta) = \frac{\text{Count}(\beta\alpha)}{\text{Count}(\beta)},$$

Code:

```python
import numpy as np
from collections import defaultdict
import matplotlib.pyplot as plt

# Load the text file and convert it to a numpy array of characters
wap_path = "war_and_peace.txt"
war_and_peace = ""
with open(wap_path, 'r', encoding='utf-8') as file:
    war_and_peace = file.read()
    # Clean the text, removing newlines and make it lowercase
    war_and_peace = war_and_peace.replace("\n", " ").replace("\r",
    ↪  "").lower()
wap_np = np.array(list(war_and_peace))


# Load the list of symbols and convert it to a numpy array
symbols_file_path = "symbols.txt"
symbols = []
with open(symbols_file_path, 'r', encoding='utf-8') as sym_file:
    symbols = [line.strip() for line in sym_file.readlines()]
    # Second symbol is the space character
    symbols[1] = " "
symbols_np = np.array(symbols)


# Create a dictionary to store the count of each symbol and symbol pair
# and iterate over the text to count the occurrences of each symbol and
↪  symbol pair
symbols_count = defaultdict(int)
symbol_pairs_count = defaultdict(int)
n = len(wap_np)
for i in range(n - 1):
    current_pair = (wap_np[i], wap_np[i + 1])
    # Ensure current pair is valid (exists in symbols list)
    if current_pair[0] in symbols and current_pair[1] in symbols:
        symbols_count[current_pair[0]] += 1
        symbol_pairs_count[current_pair] += 1
# Edge case, count the last symbol
if wap_np[-1] in symbols:
    symbols_count[wap_np[-1]] += 1


# Compute the transition probability matrix and marginal probability vector
num_symbols = len(symbols)
transition_matrix = np.zeros((num_symbols, num_symbols))
```

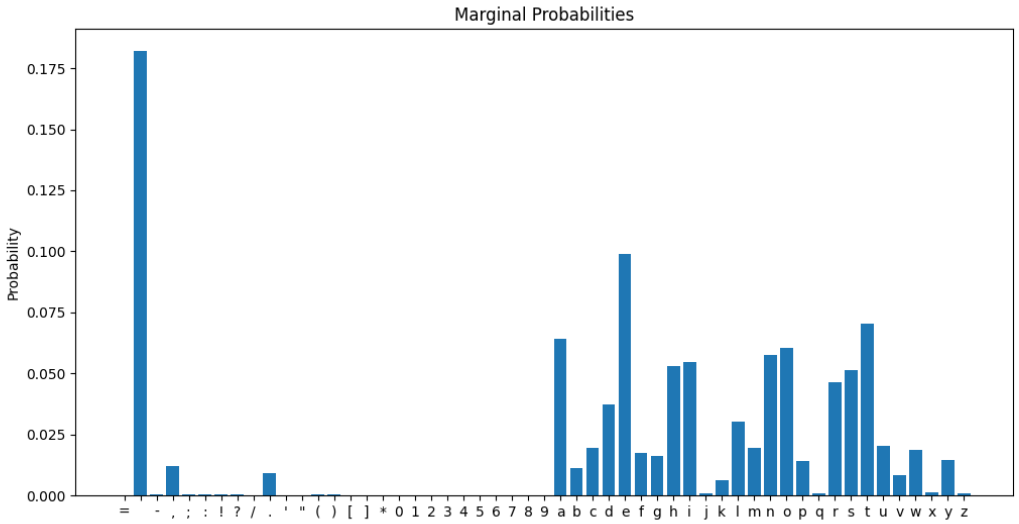
```
marginal_probabilities = np.zeros(num_symbols)
# Fill in the transition matrix with counts of specific symbol pairs
for current_pair, count in symbol_pairs_count.items():
    i = symbols.index(current_pair[0])
    j = symbols.index(current_pair[1])
    transition_matrix[i, j] = count
# Normalize the rows of the transition matrix to get probabilities
for i in range(num_symbols):
    row_sum = sum(transition_matrix[i, :])
    if row_sum > 0:
        transition_matrix[i, :] /= row_sum
# Fill marginal probabilities vector
total_symbols_count = sum(symbols_count.values())
for symbol, count in symbols_count.items():
    i = symbols.index(symbol)
    marginal_probabilities[i] = count / total_symbols_count
```

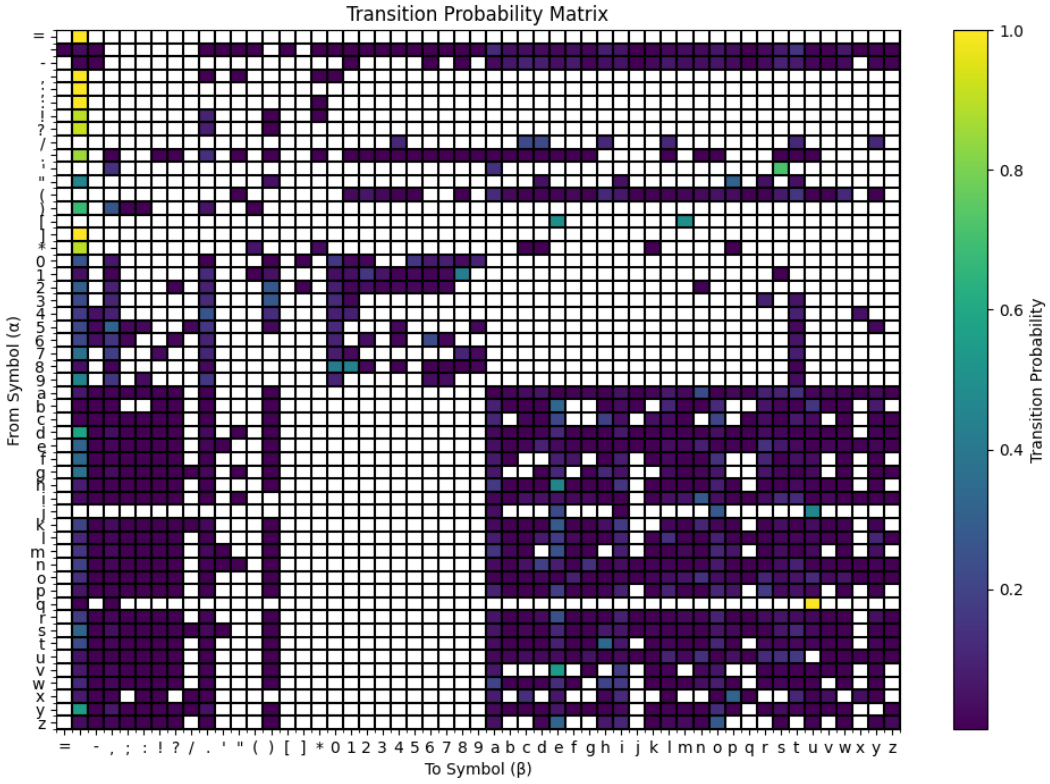
Plot Our Transition and Marginal Probabilities:

```
# Create a custom colormap where masked (zero) values are displayed as white
masked_matrix = np.ma.masked_where(transition_matrix == 0, transition_matrix)
cmap = plt.cm.viridis
cmap.set_bad(color='white')
# Plot the transition matrix with custom colormap and gridlines
plt.figure(figsize=(12, 8))
plt.imshow(masked_matrix, cmap=cmap, aspect='auto')
plt.colorbar(label="Transition Probability")
plt.grid(which='minor', color='black', linestyle='-', linewidth=1.5)
plt.xticks(ticks=np.arange(num_symbols), labels=symbols)
plt.yticks(ticks=np.arange(num_symbols), labels=symbols)
plt.gca().set_xticks(np.arange(-.5, num_symbols, 1), minor=True)
plt.gca().set_yticks(np.arange(-.5, num_symbols, 1), minor=True)
plt.title("Transition Probability Matrix")
plt.xlabel("To Symbol ()")
plt.ylabel("From Symbol ()")
plt.savefig("q5a_transition_prob")


# Plot the marginal probabilities as a bar plot
plt.figure(figsize=(12, 6))
plt.bar(np.arange(num_symbols), marginal_probabilities)
plt.xticks(ticks=np.arange(num_symbols), labels=symbols)
plt.title("Marginal Probabilities")
plt.ylabel("Probability")
plt.savefig("q5a_marginal_prob")
```

To find the printed values from these plots, please check the appendix. To summarise this

information, though, I thought it would be much nicer to display these values visually. It is important to note that the white values in the transition probability matrix represent the values being $p(S_i = \alpha \mid S_{i-1} = \beta) = 0$.





5.2 Part B: Write down the joint probability of the encrypted text

Using the knowledge that every encrypted symbol $e_i = \sigma(s_i)$, we can therefore make the assumption that there is an inverse mapping (since the mapping is one-to-one) that says $s_i = \sigma^{-1}(e_i)$. Using our prerequisite knowledge, we can therefore say that...

$$p(e_1, e_2, ..., e_n \mid \sigma) = p(\sigma^{-1}(e_1)) \prod_{i=2}^{n} p(\sigma^{-1}(e_i)) \cdot p(\sigma^{-1}(e_i) \mid \sigma^{-1}(e_{i-1}))$$

5.3 Part C: Metropolis-Hastings Algorithm

MCMC is a very general and powerful framework for sampling that works well across high-dimensional space. In our scenario, we are going to use the Metropolis-Hastings algorithm. This algorithm can be roughly laid out in the general following way...

Algorithm 1 The Metropolis Algorithm (General Form)

Require: $\pi(x)$ be a known probability distribution

$x_t \leftarrow x_0$ ▷ Initial value of chain
$t \leftarrow 0$
for $t = 1, 2, \ldots$ **do**
    Generate a candidate x' from proposal distribution $q(x \mid y)$

    Calculate the acceptance probability $\mathrm{A}(x, x') = \min\left(1, \frac{\pi(x')q(x_t|x')}{\pi(x_t)q(x'|x_t)}\right)$
    Generate $u \sim \mathcal{U}(0, 1)$
    if $u \leq \alpha$ **then**
        Accept x' and set $x_{t+1} = x'$
    else
        Reject x' and set $x_{t+1} = x_t$
    end if
end for
return $(x_0, ..., x_t)$

Our proposal is given, by using a given initial mapping σ, and randomly switching the mapping of one symbol to be the mapping of another symbol, and vice versa. This implies that our proposal distribution is symmetric and random, $\therefore S(\sigma \to \sigma') = S(\sigma' \to \sigma)$. This in fact simplifies our algorithm greatly, to become the simpler Metropolis algorithm. Therefore, the acceptance probability simplifies...

$$\mathrm{A}(x, x') = \min\left(1, \frac{\pi(x') \cdot S(\sigma \to \sigma')}{\pi(x) \cdot S(\sigma' \to \sigma)}\right)$$

$$\implies \mathrm{A}(x, x') = \min\left(1, \frac{\pi(x')}{\pi(x)}\right)$$

In our case, the target distribution, is the joint probability of the encrypted text given the mapping σ. Therefore...

$$\implies \mathrm{A}(\sigma, \sigma') = \min\left(1, \frac{p(e_1, e_2, ..., e_n \mid \sigma')}{p(e_1, e_2, ..., e_n \mid \sigma)}\right)$$

5.4 Part D: Implement the Metropolis-Hastings Algorithm

In order to correctly and effectively implement the algorithm, we are going to make some adjustments, these are mostly to prevent numerical underflow...

1. **Log Domain:** Due to the probability values we are dealing with being so small, it is more computationally correct to work in the $\log(\dots)$ domain, we will therefore always work with the logarithmic versions of our formulas and algorithms.
2. **Ergodicity:** Return ergodicity to the chain by adding a small constant to every term in the transition probability matrix, referred to as "alpha" in the "log_likelihood" function.

Code:

```python
import numpy as np
import collections

# Load the encrypted message and transform to numpy array
message_path = "message.txt"
message: str
with open(message_path, 'r', encoding='utf-8') as file:
    message = file.read()
message = message.replace("\n", " ").replace("\r", "").lower()
message_np = np.array(list(message))


def decrypt_message(sigma, message=message, symbols=symbols):
    """Decrypt the message using the given mapping sigma."""
    decrypted_text = np.array([sigma[symbols.index(c)] for c in message])
    return "".join(decrypted_text)


def log_likelihood(
        transition_matrix,
        marginal_probabilities,
        sigma,
        symbols=symbols,
    ):
    """Calculate the likelihood of the encrypted text given the mapping
    sigma from the unencrypted text to the encrypted text. We have to
    reverse the mapping to get the likelihood of the unencrypted text.
    """
    # Map the encrypted text to the unencrypted text using sigma
    decrypted_text = decrypt_message(sigma)

    # To prevent numerical issues, we add a small constant to the transition
    # matrix, this acts to restore ergodicity to the chain.
    alpha = 1e-2

    # Initialize the likelihood
    log_likelihood =
    ↪  np.log(marginal_probabilities[symbols.index(decrypted_text[0])])
    # print(f'Log likelihood 1st term: {log_likelihood}')
    for i in range(1, len(decrypted_text)):
        # Get the transition probability from the previous symbol to the
        ↪  current symbol
        prev_symbol = decrypted_text[i - 1]
        current_symbol = decrypted_text[i]
```

```
        log_likelihood +=
        ↪  np.log(transition_matrix[symbols.index(prev_symbol),
        ↪  symbols.index(current_symbol)] + alpha)
        # print(f'Log likelihood {i+1}th term: {log_likelihood}')

    return log_likelihood


def calculate_log_acceptance_probability(
        sigma,
        sigma_proposed,
    ):
    """Calculate the log acceptance probability for the proposed state, using
    ↪  the
    formula: min(0, log_likelihood_proposed - log_likelihood_current).
    """
    # Calculate the likelihood of the proposed state
    log_likelihood_proposed = log_likelihood(
        transition_matrix,
        marginal_probabilities,
        sigma_proposed,
    )
    # Calculate the likelihood of the current state
    log_likelihood_current = log_likelihood(
        transition_matrix,
        marginal_probabilities,
        sigma,
    )
    return min(0, log_likelihood_proposed - log_likelihood_current)


def sigma_prpoosal(
        sigma,
    ):
    """ Generate a new proposal for the MCMC algorithm, by swapping two
    ↪  symbols in sigma."""
    sigma_proposed = sigma.copy()
    i, j = np.random.choice(len(sigma), 2, replace=False)
    sigma_proposed[i], sigma_proposed[j] = sigma_proposed[j],
    ↪  sigma_proposed[i]
    return sigma_proposed


def initialize_sigma(
        symbols: list=symbols,
        encrypted_text=message_np
    ):
```

```
    """Given a message as a numpy array of characters, initialize the mapping sigma. Sigma
    is a permutation of the symbols list, which maps the symbols in the encrypted text to the
    symbols in the unencrypted text. We can use the fact that the space character is the most
    common symbol in the English language to initialize sigma. We will then find the most common
    symbol in the encrypted text and map it to the space character.

    Args:
    symbols: list of symbols
    encrypted_text: numpy array of characters representing the encrypted message

    Returns:
    sigma_init: numpy array of characters representing the initial mapping
    """
    # Initialize the mapping sigma
    sigma_init = np.random.permutation(symbols)

    # Find the most common symbol in the encrypted text
    counter = collections.Counter(encrypted_text)
    most_common_symbol = counter.most_common(1)[0][0]

    # Find the index of the most common symbol in the symbols list
    most_common_symbol_index = symbols.index(most_common_symbol)

    # # Find the index of the space character in the symbols list
    # space_index = symbols.index(" ")

    # Swap the most common symbol with the space character in the mapping sigma
    swap_letter = sigma_init[most_common_symbol_index]
    sigma_init[np.where(sigma_init == " ")[0]] = swap_letter
    sigma_init[most_common_symbol_index] = " "

    # # Swap the space character with the most common symbol in the mapping sigma
    # sigma_init[space_index] = most_common_symbol

    return sigma_init


def run_mcmc(
        N_iters: int,
        symbols: list,
```

```
        smart_sigma: bool=None,
        encrypted_text: np.array=message_np,
        transition_matrix: np.array=transition_matrix,
        marginal_probabilities: np.array=marginal_probabilities
    ):
    """Run an MCMC algorithm for a fixed number of iterations on our
    encrypted text. We will use the Metropolis-Hastings algorithm to
    sample from the space of possible mappings sigma. We will use the
    log likelihood of the encrypted text as the target distribution.
    Each proposal though, only swaps two symbols in current sigma,
    to produce our new 'proposal' sigma. We will accept the proposal
    with the appropriate probability, based on the ratio of the
    log-likelihoods of the proposed and current states. We will
    print the decrypted text every 100 iterations, along with
    the likelihood of the current state.

    To initialize intelligently, we can use the fact that the space
    character is the most common, and initialize the mapping sigma
    accordingly, with the smart_sigma flag.

    Inputs:
    N_iters: int, number of iterations to run the MCMC algorithm
    symbols: list of symbols
    smart_sigma: bool, flag to initialize the mapping sigma intelligently
    encrypted_text: numpy array of chars representing the encrypted message
    transition_matrix: numpy array representing the transition matrix
    marginal_probabilities: numpy array of the marginal probabilities

    Returns:
    sigma: numpy array of characters representing the final mapping
    """
    if not smart_sigma: sigma = np.random.permutation(symbols)
    if smart_sigma: sigma = initialize_sigma()
    N_accepted = 0
    i = 0
    for i in range(N_iters):
        # Initialize the acceptance flag and generate a proposal
        accepted, sigma_proposed = False, sigma_prpoosal(sigma)
        # Accept the proposal with the appropriate probability
        log_acceptance_probability = calculate_log_acceptance_probability(
            sigma,
            sigma_proposed,
        )
        if np.log(np.random.rand()) < log_acceptance_probability:
            sigma = sigma_proposed
            N_accepted += 1
            accepted = True
```

```
        # Print the decrypted text every 100 iterations
        if i % 100 == 0:
            decrypted_text = decrypt_message(sigma)
            decrypted_text_str = "".join(decrypted_text)
            print(f'Decrypted Text (Iter = {i}):
            ↪  "{decrypted_text_str[:60]}"')

            # Likelihood of the current state
            likelihood_current = log_likelihood(
                transition_matrix,
                marginal_probabilities,
                sigma,
            )
            print(f'Likelihood (Iter = {i}): {likelihood_current:.2e}')
    return sigma


# Run the MCMC algorithm
N_iters = 10000
sigma = run_mcmc(N_iters, symbols, smart_sigma=True)

# Decrypt the message using the final mapping sigma
decrypted_message = decrypt_message(sigma)
print(f'FINAL DECRYPTED MESSAGE: {decrypted_message}')
```

Results:

```
Decrypted Text (Iter = 0): "t, kl l*v,m/j c,2 k*j/ )v1,/jcx1/ l/cjp kl sc'0/j
↪  mc)/ k/ p*"
Likelihood (Iter = 0): -1.38e+04
Decrypted Text (Iter = 100): "a, nf f1d,.p" c,2 n1"p mdo,p"chop fpc"i nf
↪  8ctsp" .cmp np i1"
Likelihood (Iter = 100): -8.38e+03
Decrypted Text (Iter = 200): "a, 1r rnd,p.h c,2 1nh. mdo,.hcbo. r.chi 1r
↪  !cts.h pcm. 1. in"
Likelihood (Iter = 200): -7.73e+03
Decrypted Text (Iter = 300): "s- mr rnd-p.a c-2 mna. 1do-.acbo. r.cai mr
↪  !cth.a pc1. m. in"
Likelihood (Iter = 300): -7.29e+03
Decrypted Text (Iter = 400): "s- kr rnd-m.a c-2 kna. 1do-.acfo. r.cai kr
↪  ycth.a mc1. k. in"
Likelihood (Iter = 400): -6.94e+03
Decrypted Text (Iter = 500): "s- km mld-r.a c-f kla. 1do-.ac2o. m.cai km
↪  ycth.a rc1. k. il"
Likelihood (Iter = 500): -6.79e+03
Decrypted Text (Iter = 600): "s- km mld-era n-f klar wdo-ranpor mrnai km
↪  ynthra enwr kr il"
```

```
Likelihood (Iter = 600): -5.87e+03
Decrypted Text (Iter = 700): "sk fm mldkira nkw flar -dokranpor mrnae fm
↪  ynthra in-r fr el"
Likelihood (Iter = 700): -5.54e+03
Decrypted Text (Iter = 800): "sk cm mlykira nkf clar uyokranpor mrnae cm
↪  wnthra inur cr el"
Likelihood (Iter = 800): -5.41e+03
Decrypted Text (Iter = 900): "sk cm mlikyra nkf clar uiokranpor mrnae cm
↪  wnthra ynur cr el"
Likelihood (Iter = 900): -5.37e+03
Decrypted Text (Iter = 1000): "sk cm mlikyra nkf clar uiokran-or mrnae cm
↪  wnthra ynur cr el"
Likelihood (Iter = 1000): -5.29e+03
Decrypted Text (Iter = 1100): "sm ck klimfra nmy clar uiomran-or krnae ck
↪  wnthra fnur cr el"
Likelihood (Iter = 1100): -5.22e+03
Decrypted Text (Iter = 1200): "sm ck klnmfra imy clar unomrai-or kriae ck
↪  withra fiur cr el"
Likelihood (Iter = 1200): -5.09e+03
Decrypted Text (Iter = 1300): "sa ck klnafrm iay clmr unoarmi-or krime ck
↪  githrm fiur cr el"
Likelihood (Iter = 1300): -5.02e+03
Decrypted Text (Iter = 1400): "sa ck konamrf iay cofr unlarfi-lr krife ck
↪  githrf miur cr eo"
Likelihood (Iter = 1400): -4.96e+03
Decrypted Text (Iter = 1500): "so ck kanomrf ioy cafr unlorfivlr krife ck
↪  githrf miur cr ea"
Likelihood (Iter = 1500): -4.93e+03
Decrypted Text (Iter = 1600): "uo cy yinomef aov cife snloefakle yeafr cy
↪  gathef mase ce ri"
Likelihood (Iter = 1600): -4.73e+03
Decrypted Text (Iter = 1700): "ur cy yinrkef arv cife snlrefamle yeafo cy
↪  gathef kase ce oi"
Likelihood (Iter = 1700): -4.62e+03
Decrypted Text (Iter = 1800): "us cy yinskef asg cife rnlsefable yeafo cy
↪  vathef kare ce oi"
Likelihood (Iter = 1800): -4.56e+03
Decrypted Text (Iter = 1900): "us fy yinskec asg fice rnlsecable yeaco fy
↪  vathec kare fe oi"
Likelihood (Iter = 1900): -4.55e+03
Decrypted Text (Iter = 2000): "us fy yinskec asg fice rnlsecable yeaco fy
↪  vathec kare fe oi"
Likelihood (Iter = 2000): -4.55e+03
Decrypted Text (Iter = 2100): "us fy yinskec asg fice rnlsecable yeaco fy
↪  vathec kare fe oi"
Likelihood (Iter = 2100): -4.55e+03
```

```
Decrypted Text (Iter = 2200): "us fy yiosked asg fide rolsedable yeadn fy vathed kare fe ni"
Likelihood (Iter = 2200): -4.42e+03
Decrypted Text (Iter = 2300): "us fy yiosked asl fide gorsedabre yeadn fy mathed kage fe ni"
Likelihood (Iter = 2300): -4.36e+03
Decrypted Text (Iter = 2400): "us fy yiosked asl fide gorsedabre yeadn fy mathed kage fe ni"
Likelihood (Iter = 2400): -4.34e+03
Decrypted Text (Iter = 2500): "us fy yiosked asl fide gorsedabre yeadn fy mathed kage fe ni"
Likelihood (Iter = 2500): -4.34e+03
Decrypted Text (Iter = 2600): "us fy yiosked asl fide gorsedabre yeadn fy mathed kage fe ni"
Likelihood (Iter = 2600): -4.34e+03
Decrypted Text (Iter = 2700): "us by yioskem asl bime gorsemafre yeamn by dathem kage be ni"
Likelihood (Iter = 2700): -4.34e+03
Decrypted Text (Iter = 2800): "un by yionkem anl bime gornemafre yeams by dathem kage be si"
Likelihood (Iter = 2800): -4.21e+03
Decrypted Text (Iter = 2900): "un by yiongem anl bime kornemafre yeams by dathem gake be si"
Likelihood (Iter = 2900): -4.19e+03
Decrypted Text (Iter = 3000): "un by yiongem anl bime kornemafre yeams by dathem gake be si"
Likelihood (Iter = 3000): -4.15e+03
Decrypted Text (Iter = 3100): "un by yionger anl bire komnerafme years by dather gake be si"
Likelihood (Iter = 3100): -4.13e+03
Decrypted Text (Iter = 3200): "un by yionger anl bire komnerafme years by dather gake be si"
Likelihood (Iter = 3200): -4.13e+03
Decrypted Text (Iter = 3300): "un by yionger and bire komnerapme years by lather gake be si"
Likelihood (Iter = 3300): -4.09e+03
Decrypted Text (Iter = 3400): "un by yoinger and bore kimnerapme years by lather gake be so"
Likelihood (Iter = 3400): -4.08e+03
Decrypted Text (Iter = 3500): "un by yoinger and bore kimneralme years by pather gake be so"
Likelihood (Iter = 3500): -4.07e+03
Decrypted Text (Iter = 3600): "un by yoinger and bore vilneramle years by pather gave be so"
Likelihood (Iter = 3600): -3.97e+03
Decrypted Text (Iter = 3700): "un by yoinger and bore vilneramle years by pather gave be so"
```

```
Likelihood (Iter = 3700): -3.97e+03
Decrypted Text (Iter = 3800): "un by yoinger and bore vilneramle years by
↪  pather gave be so"
Likelihood (Iter = 3800): -3.97e+03
Decrypted Text (Iter = 3900): "un by yoinger and bore pilneramle years by
↪  vather gape be so"
Likelihood (Iter = 3900): -3.97e+03
Decrypted Text (Iter = 4000): "un by yoinger and bore pilneramle years by
↪  vather gape be so"
Likelihood (Iter = 4000): -3.97e+03
Decrypted Text (Iter = 4100): "un by yoinger and bore pilneramle years by
↪  vather gape be so"
Likelihood (Iter = 4100): -3.96e+03
Decrypted Text (Iter = 4200): "un by yoinger and bore pilneramle years by
↪  vather gape be so"
Likelihood (Iter = 4200): -3.96e+03
Decrypted Text (Iter = 4300): "un by yoinger and bore pilneramle years by
↪  vather gape be so"
Likelihood (Iter = 4300): -3.96e+03
Decrypted Text (Iter = 4400): "in by younger and bore pulneramle years by
↪  vather gape be so"
Likelihood (Iter = 4400): -3.79e+03
Decrypted Text (Iter = 4500): "in by younger and bore pulneramle years by
↪  vather gape be so"
Likelihood (Iter = 4500): -3.79e+03
Decrypted Text (Iter = 4600): "in by younger and bore vulneramle years by
↪  pather gave be so"
Likelihood (Iter = 4600): -3.76e+03
Decrypted Text (Iter = 4700): "in by younger and bore vulneramle years by
↪  pather gave be so"
Likelihood (Iter = 4700): -3.76e+03
Decrypted Text (Iter = 4800): "in by younger and bore vulneramle years by
↪  pather gave be so"
Likelihood (Iter = 4800): -3.75e+03
Decrypted Text (Iter = 4900): "in by younger and bore vulneramle years by
↪  pather gave be so"
Likelihood (Iter = 4900): -3.75e+03
Decrypted Text (Iter = 5000): "in by younger and bore vulneramle years by
↪  pather gave be so"
Likelihood (Iter = 5000): -3.75e+03
Decrypted Text (Iter = 5100): "in by younger and bore vulneramle years by
↪  pather gave be so"
Likelihood (Iter = 5100): -3.75e+03
Decrypted Text (Iter = 5200): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 5200): -3.68e+03
```

```
Decrypted Text (Iter = 5300): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 5300): -3.68e+03
Decrypted Text (Iter = 5400): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 5400): -3.68e+03
Decrypted Text (Iter = 5500): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 5500): -3.68e+03
Decrypted Text (Iter = 5600): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 5600): -3.68e+03
Decrypted Text (Iter = 5700): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 5700): -3.68e+03
Decrypted Text (Iter = 5800): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 5800): -3.68e+03
Decrypted Text (Iter = 5900): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 5900): -3.68e+03
Decrypted Text (Iter = 6000): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 6000): -3.68e+03
Decrypted Text (Iter = 6100): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 6100): -3.68e+03
Decrypted Text (Iter = 6200): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 6200): -3.68e+03
Decrypted Text (Iter = 6300): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 6300): -3.68e+03
Decrypted Text (Iter = 6400): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 6400): -3.66e+03
Decrypted Text (Iter = 6500): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 6500): -3.66e+03
Decrypted Text (Iter = 6600): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 6600): -3.66e+03
Decrypted Text (Iter = 6700): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 6700): -3.66e+03
Decrypted Text (Iter = 6800): "in my younger and more vulnerable years my
↪  pather gave me so"
```

```
Likelihood (Iter = 6800): -3.66e+03
Decrypted Text (Iter = 6900): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 6900): -3.66e+03
Decrypted Text (Iter = 7000): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 7000): -3.66e+03
Decrypted Text (Iter = 7100): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 7100): -3.66e+03
Decrypted Text (Iter = 7200): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 7200): -3.66e+03
Decrypted Text (Iter = 7300): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 7300): -3.66e+03
Decrypted Text (Iter = 7400): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 7400): -3.66e+03
Decrypted Text (Iter = 7500): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 7500): -3.66e+03
Decrypted Text (Iter = 7600): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 7600): -3.66e+03
Decrypted Text (Iter = 7700): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 7700): -3.66e+03
Decrypted Text (Iter = 7800): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 7800): -3.66e+03
Decrypted Text (Iter = 7900): "in my younger and more vulnerable years my
↪  pather gave me so"
Likelihood (Iter = 7900): -3.66e+03
Decrypted Text (Iter = 8000): "in my younger and more vulnerable years my
↪  father gave me so"
Likelihood (Iter = 8000): -3.62e+03
Decrypted Text (Iter = 8100): "in my younger and more vulnerable years my
↪  father gave me so"
Likelihood (Iter = 8100): -3.62e+03
Decrypted Text (Iter = 8200): "in my younger and more vulnerable years my
↪  father gave me so"
Likelihood (Iter = 8200): -3.62e+03
Decrypted Text (Iter = 8300): "in my younger and more vulnerable years my
↪  father gave me so"
Likelihood (Iter = 8300): -3.61e+03
```

```
Decrypted Text (Iter = 8400): "in my younger and more vulnerable years my father gave me so"
Likelihood (Iter = 8400): -3.61e+03
Decrypted Text (Iter = 8500): "in my younger and more vulnerable years my father gave me so"
Likelihood (Iter = 8500): -3.61e+03
Decrypted Text (Iter = 8600): "in my younger and more vulnerable years my father gave me so"
Likelihood (Iter = 8600): -3.61e+03
Decrypted Text (Iter = 8700): "in my younger and more vulnerable years my father gave me so"
Likelihood (Iter = 8700): -3.61e+03
Decrypted Text (Iter = 8800): "in my younger and more vulnerable years my father gave me so"
Likelihood (Iter = 8800): -3.61e+03
Decrypted Text (Iter = 8900): "in my younger and more vulnerable years my father gave me so"
Likelihood (Iter = 8900): -3.61e+03
Decrypted Text (Iter = 9000): "in my younger and more vulnerable years my father gave me so"
Likelihood (Iter = 9000): -3.61e+03
Decrypted Text (Iter = 9100): "in my younger and more vulnerable years my father gave me so"
Likelihood (Iter = 9100): -3.61e+03
Decrypted Text (Iter = 9200): "in my younger and more vulnerable years my father gave me so"
Likelihood (Iter = 9200): -3.61e+03
Decrypted Text (Iter = 9300): "in my younger and more vulnerable years my father gave me so"
Likelihood (Iter = 9300): -3.61e+03
Decrypted Text (Iter = 9400): "in my younger and more vulnerable years my father gave me so"
Likelihood (Iter = 9400): -3.61e+03
Decrypted Text (Iter = 9500): "in my younger and more vulnerable years my father gave me so"
Likelihood (Iter = 9500): -3.61e+03
Decrypted Text (Iter = 9600): "in my younger and more vulnerable years my father gave me so"
Likelihood (Iter = 9600): -3.61e+03
Decrypted Text (Iter = 9700): "in my younger and more vulnerable years my father gave me so"
Likelihood (Iter = 9700): -3.61e+03
Decrypted Text (Iter = 9800): "in my younger and more vulnerable years my father gave me so"
Likelihood (Iter = 9800): -3.61e+03
Decrypted Text (Iter = 9900): "in my younger and more vulnerable years my father gave me so"
```

```
Likelihood (Iter = 9900): -3.61e+03
FINAL DECRYPTED MESSAGE: in my younger and more vulnerable years my father gave me some advice that ixve been turning over in my mind ever since! *whenever you feel like criticizing any one,* he told me, *just remember that all the people in this world havenxt had the advantages that youxve had!* he didnxt say any more but wexve always been unusually communicative in a reserved way, and i understood that he meant a great deal more than that! in consequence ixm inclined to reserve all judgments, a habit that has opened up many curious natures to me and also made me the victim of not a few veteran bores! the abnormal mind is quick to detect and attach itself to this quality when it appears in a normal person, and so it came about that in college i was unjustly accused of being a politician, because i was privy to the secret griefs of wild, unknown men! most of the confidences were unsought..frequently i have feigned sleep, preoccupation, or a hostile levity when i realized by some unmistakable sign that an intimate revelation was quivering on the horizon..for the intimate revelations of young men or at least the terms in which they e-press them are usually plagiaristic and marred by obvious suppressions! reserving judgments is a matter of infinite hope! i am still a little afraid of missing something if i forget that, as my father snobbishly suggested, and i snobbishly repeat a sense of the fundamental decencies is parcelled out unequally at birth!
```

5.5 Part E: Ergodicity of the Chain

Saying that a Markov chain is ergodic implies that transitioning from one state or another state is possible in a finite number of steps. One of the main implications of a Markov chain being ergodic is that after sufficient burn-in of the chain, we always converge to the same unique stationary distribution, independent of the initial state of the chain.

"Does this affect the ergodicity of the chain?"
Yes, the ergodicity of the chain is affected. To restore ergodicity, as previously stated in part D, we must try and remove/replace our zero-valued elements of the transition matrix, with some non-zero values. To do this, we will simply add a very small constant "alpha" to all elements in the transition matrix.

This also helps us to efficiently calculate the likelihood for a given mapping since otherwise, due to some elements being null, we would have to compute $\log(0)$ which for the computer is negative infinity. This also represents a likely more accurate transition matrix, since it is hard to think of many situations in English where the transition probability between two symbols is 0.

5.6 Part F: Further Qualitative Analysis

For instance, would symbol probabilities alone (rather than transitions) be sufficient?
No, symbol probabilities alone would not be sufficient. By not considering the probabilities of the order of symbols, or in other words the transition probabilities, we would be missing out on a lot of information which is very important for decryption.

If we used a second-order Markov chain for English text, what problems might we encounter?
A second-order Markov chain would mean that we would now have to consider the previous 2 symbols before the current symbol. This would mean that we would now have a transition tensor of $53 \times 53 \times 53$. It is likely we would see an increase in the accuracy of our decryption mappings but at an increased computational cost. It would also be likely that to accurately use this model and create a good transition tensor we would need to sample from a much larger body, or multiple bodies, of text. In all likelihood, though, a model like this would not be needed for decrypting English messages.

Will it work if the encryption scheme allows two symbols to be mapped to the same encrypted value?
The current model would break if the mapping was not one-to-one. We would need to derive the formulas for calculating log-likelihood under this new assumption, as well as make adjustments to how we are going to generate new proposals, with any other side effects. The current formulas would not work optimally, and therefore neither the model.

Would it work for Chinese with more than 10000 symbols?
The first-order Markov chain would likely have to be much more fine-tuned to be able to remotely work on Chinese. Given we would have many more symbols and a much larger transition matrix, we would need to collect an immense out of Chinese text to get mostly non-zero elements in the transition matrix. The size of the number of symbols and quite large computational costs lead me to believe that while it could be theoretically possible to do so, it may not be practically feasible or intelligent. It is likely that there is a better way to model this kind of encryption.

6 Implementing Gibbs sampling for LDA (60 bonus)

NOT ATTEMPTED

7 Optimization (15 marks)

7.1 Part A: Find Extrema

In this section, we want to find the extrema of the function...

$$f(x,y) := x + 2y$$

subject to the following...

$$\text{Constraint:} \quad y^2 + xy = 1.$$

To derive the solution, we will use a Lagrange multiplier method...

$$\nabla f(x,y) + \lambda \nabla g(x,y) = \vec{0} \quad \text{and} \quad \lambda \neq 0$$

where...

$$g(x,y) = y^2 + xy - 1 = 0.$$

Using the definition of the ∇ operator, we know that...

$$\nabla h(x,y) = \frac{\partial h}{\partial x}\hat{i} + \frac{\partial h}{\partial y}\hat{j}$$

Leading us to compute that

$$\therefore \nabla f(x,y) = \hat{i} + 2\hat{j}$$

and...

$$\therefore \nabla g(x,y) = y\hat{i} + (2y + x)\hat{j}$$

Combining these equations together, we get...

$$\therefore \begin{pmatrix} 1 + \lambda y \\ 2 + \lambda(2y + x) \end{pmatrix} = 0$$

Our problem has now been simplified to solving a system of 3 equations...

$$1 + \lambda y = 0 \quad (1)$$

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

$$y^2 + xy - 1 = 0 \quad (3)$$

Let us work through this...

$$(2) - 2 \cdot (1)$$

$$\implies 2 + 2\lambda y + \lambda x - 2 - 2\lambda y = 0$$

$$\therefore \lambda x = 0 \implies x = 0 \quad \text{since} \quad \lambda \neq 0$$

Plugging this result back into equation 1

$$\therefore y = -\frac{1}{\lambda}$$

Putting this back into equation 3

$$\therefore \left(-\frac{1}{\lambda}\right)^2 - 1 = 0$$

$$\implies \lambda = \pm 1$$

$$\therefore (0, 1), (0, -1) \text{ are solutions}$$

7.2 Part B: Newton's Method

- We have a routine to evaluate $\exp(x)$.
- We want to compute $\ln(a)$, $a \in \mathbb{R}_+$.
- We want to use Newton's Method.
  - Newton's Method searches for roots of f, solving $f(x) = 0$ by repeatedly iterating

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

Part I:

We know that $f(x, a) = 0$ when $x = \ln(a)$. It is logical that then to compute a we must solve $e^x = a$ or reformulated...

$$e^x - a = 0$$

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

Part II:
Let us find the update equation...

$$f(x, a) = e^x - a \implies \frac{\partial f(x, a)}{\partial x} = e^x$$

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

8 Eigenvalues as solutions of an optimization problem (20 bonus)

8.1 Part A: Show $\sup_{x\in\mathbb{R}^n} R_A(x) = \sup_{\hat{x}\in S} R_A(\hat{x})$

To prove that $\sup_{x\in\mathbb{R}^n} R_A(x)$ is attained, we begin by transforming the problem to an equivalent one over a compact domain S.

We want to show that optimizing $R_A(x)$ over $\mathbb{R}^n$ is equivalent to optimizing over S.

For any $x \in \mathbb{R}^n$, we can express x in terms of a unit vector by writing

$$x = \left(\frac{x}{\|x\|}\right) \cdot \|x\|,$$

where $\frac{x}{\|x\|}$ has unit norm. Letting $\hat{x} = \frac{x}{\|x\|}$, which satisfies $\|\hat{x}\| = 1$, we then have

$$R_A(x) = \frac{x^T A x}{x^T x} = \frac{(\|x\|\hat{x})^T A(\|x\|\hat{x})}{(\|x\|\hat{x})^T(\|x\|\hat{x})} = \frac{\|x\|^2 \hat{x}^T A \hat{x}}{\|x\|^2 \hat{x}^T \hat{x}} = \hat{x}^T A \hat{x} = R_A(\hat{x}).$$

Therefore, $R_A(x) = R_A(\hat{x})$, where $\hat{x}$ is the normalized version of x, and therefore we can write

$$\sup_{x\in\mathbb{R}^n} R_A(x) = \sup_{\hat{x}\in S} R_A(\hat{x}).$$

Now we can apply the extreme value theorem. Since R_A is continuous and S is compact, $\sup_{\hat{x}\in S} R_A(\hat{x})$ is guaranteed to be attained on S. Because we have shown that the supremum over $\mathbb{R}^n$ is equivalent to the supremum over S, it follows that $\sup_{x\in\mathbb{R}^n} R_A(x)$ is also attained.

8.2 Part B: Show that $R_A(x) \leq \lambda_1$

To begin, we use the given orthonormal basis (ONB) representation of any vector $x \in \mathbb{R}^n$. Specifically, we can write:

$$x = \sum_i (\xi_i^T x)\xi_i$$

where $\{\xi_i\}$ are orthonormal eigenvectors of A.

Next, we consider the problem for $\hat{x}$ on the unit sphere (i.e., $\|\hat{x}\| = 1$), as this is known to be equivalent from part A. Now we need to show that $R_A(\hat{x}) \leq \lambda_1$, where

$$R_A(x) = \frac{x^T A x}{x^T x} = \hat{x}^T A \hat{x}$$

Substituting the ONB representation into $\hat{x}^T A \hat{x}$, we get

$$\hat{x}^T A \hat{x} = \left(\sum_i (\xi_i^T \hat{x})\xi_i \right)^T A \left(\sum_i (\xi_i^T \hat{x})\xi_i \right).$$

Using $A\xi_i = \lambda_i \xi_i$ and the orthonormality of $\{\xi_i\}$, this expression simplifies to

$$\hat{x}^T A \hat{x} = \sum_i \lambda_i (\xi_i^T \hat{x})^2.$$

Now, we take into account the unit sphere condition $\|\hat{x}\| = 1$, which implies

$$\sum_i (\xi_i^T \hat{x})^2 = 1.$$

Thus, we can write

$$R_A(\hat{x}) = \sum_i \lambda_i (\xi_i^T \hat{x})^2,$$

where $\sum_i (\xi_i^T \hat{x})^2 = 1$ and $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$.

The critical insight here is to look at $R_A(x)$ as a weighted average of the eigenvalues λ_i, with weights $(\xi_i^T \hat{x})^2 \leq 1$ that sum to 1. Since λ_1 is the largest eigenvalue, it logically follows any such weighted average must satisfy $R_A(x) \leq \lambda_1$.

Therefore, we have proven that $R_A(x) \leq \lambda_1$.

8.3 Part C: x not in span$\{\xi_1, \ldots, \xi_k\}$, implies $R_A(x) < \lambda_1$

First, recall that since $\{\xi_1, \ldots, \xi_n\}$ forms an orthonormal basis (ONB), any vector $x \in \mathbb{R}^n$ can be written as...

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

Using the eigendecomposition formula, we know that...

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

Since x is not in span$\{\xi_1, \ldots, \xi_k\}$, we can split this sum as follows:

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

Similarly, for the denominator, we have

$$x^T x = \sum_{i=1}^{k} (\xi_i^T x)^2 + \sum_{i=k+1}^{n} (\xi_i^T x)^2.$$

A key observation here is that since $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ and x has non-zero components in span$\{\xi_{k+1}, \ldots, \xi_n\}$, we know that $\lambda_i < \lambda_1$ for all $i > k$.

Therefore,

$$R_A(x) = \frac{\sum_{i=1}^{k} \lambda_i (\xi_i^T x)^2 + \sum_{i=k+1}^{n} \lambda_i (\xi_i^T x)^2}{\sum_{i=1}^{n} (\xi_i^T x)^2} < \lambda_1.$$

The strict inequality holds because, at least one term in $\sum_{i=k+1}^{n} (\xi_i^T x)^2$ is non-zero (since x is not in span$\{\xi_1, \ldots, \xi_k\}$).

Therefore, $R_A(x) < \lambda_1$ for any x not contained in span$\{\xi_1, \ldots, \xi_k\}$.

Appendix

Question 5, Part A

The printed values for the marginal and transition probabilities have been generated by the code below...

Code:
This code has to be run after the initial code from question 5, part A...

```
# Print the transition matrix and marginal probabilities values
for i, symbol in enumerate(symbols):
    print(f"Symbol: {symbol}")
    print(f"Marginal Probability p('{symbol}'): {marginal_probabilities[i]}")
    print("Transition Probabilities:")
    for j, to_symbol in enumerate(symbols):
        print(f"p('{to_symbol}'|'{symbol}'): {transition_matrix[i, j]}")
    print("\n")
```

Result:

```
Symbol: =
Marginal Probability p('='): 6.34680665183418e-07
Transition Probabilities:
p('='|'='): 0.0
p(' '|'='): 1.0
p('-'|'='): 0.0
p(','|'='): 0.0
p(';'|'='): 0.0
p(':'|'='): 0.0
p('!'|'='): 0.0
p('?'|'='): 0.0
p('/'|'='): 0.0
p('.'|'='): 0.0
p('''|'='): 0.0
p('"'|'='): 0.0
p('('|'='): 0.0
p(')'|'='): 0.0
p('['|'='): 0.0
p(']'|'='): 0.0
p('*'|'='): 0.0
p('0'|'='): 0.0
p('1'|'='): 0.0
p('2'|'='): 0.0
p('3'|'='): 0.0
```

```
p('4'|'='): 0.0
p('5'|'='): 0.0
p('6'|'='): 0.0
p('7'|'='): 0.0
p('8'|'='): 0.0
p('9'|'='): 0.0
p('a'|'='): 0.0
p('b'|'='): 0.0
p('c'|'='): 0.0
p('d'|'='): 0.0
p('e'|'='): 0.0
p('f'|'='): 0.0
p('g'|'='): 0.0
p('h'|'='): 0.0
p('i'|'='): 0.0
p('j'|'='): 0.0
p('k'|'='): 0.0
p('l'|'='): 0.0
p('m'|'='): 0.0
p('n'|'='): 0.0
p('o'|'='): 0.0
p('p'|'='): 0.0
p('q'|'='): 0.0
p('r'|'='): 0.0
p('s'|'='): 0.0
p('t'|'='): 0.0
p('u'|'='): 0.0
p('v'|'='): 0.0
p('w'|'='): 0.0
p('x'|'='): 0.0
p('y'|'='): 0.0
p('z'|'='): 0.0


Symbol:
Marginal Probability p(' '): 0.18207718922781893
Transition Probabilities:
p('='|' '): 3.485784100990137e-06
p(' '|' '): 0.029043553129449822
p('-'|' '): 2.7886272807921097e-05
p(','|' '): 0.0
p(';'|' '): 0.0
p(':'|' '): 0.0
p('!'|' '): 0.0
p('?'|' '): 0.0
p('/'|' '): 0.0
p('.'|' '): 3.485784100990137e-05
```

```
p('''|' '): 1.7428920504950686e-06
p('"'|' '): 1.7428920504950685e-05
p('('|' '): 0.0011259082646198142
p(')'|' '): 0.0
p('['|' '): 3.485784100990137e-06
p(']'|' '): 0.0
p('*'|' '): 0.0004792953138861438
p('0'|' '): 3.485784100990137e-06
p('1'|' '): 0.0004426945808257474
p('2'|' '): 4.531519331287178e-05
p('3'|' '): 2.7886272807921097e-05
p('4'|' '): 1.5686028454455618e-05
p('5'|' '): 1.220024435346548e-05
p('6'|' '): 3.1372056908911235e-05
p('7'|' '): 1.220024435346548e-05
p('8'|' '): 1.3943136403960548e-05
p('9'|' '): 1.220024435346548e-05
p('a'|' '): 0.11948570741373991
p('b'|' '): 0.04295008880034997
p('c'|' '): 0.036741907316486536
p('d'|' '): 0.02952807711948745
p('e'|' '): 0.02024020538239923
p('f'|' '): 0.036935368334091494
p('g'|' '): 0.016728277900651668
p('h'|' '): 0.08569625923079202
p('i'|' '): 0.05140834392140254
p('j'|' '): 0.002818256445650526
p('k'|' '): 0.006940196145071363
p('l'|' '): 0.02269594028154678
p('m'|' '): 0.03355764354023205
p('n'|' '): 0.02548456756233889
p('o'|' '): 0.05860474519789668
p('p'|' '): 0.03264611099782313
p('q'|' '): 0.0024557348991475513
p('r'|' '): 0.026263640308910187
p('s'|' '): 0.07040761016384928
p('t'|' '): 0.15062421678788482
p('u'|' '): 0.009671307988197136
p('v'|' '): 0.006814707917435717
p('w'|' '): 0.06848345734010272
p('x'|' '): 0.0007825585306722858
p('y'|' '): 0.01043295181426348
p('z'|' '): 0.00024749067117029973


Symbol: -
Marginal Probability p('-'): 0.000580098127977644
```

```
Transition Probabilities:
p('='|'-'): 0.0
p(' '|'-'): 0.0087527352297593
p('-'|'-'): 0.002735229759299781
p(','|'-'): 0.0
p(';'|'-'): 0.0
p(':'|'-'): 0.0
p('!'|'-'): 0.0
p('?'|'-'): 0.0
p('/'|'-'): 0.0
p('.'|'-'): 0.0
p('''|'-'): 0.0
p('"'|'-'): 0.0
p('('|'-'): 0.0
p(')'|'-'): 0.0
p('['|'-'): 0.0
p(']'|'-'): 0.0
p('*'|'-'): 0.0
p('0'|'-'): 0.0
p('1'|'-'): 0.0005470459518599562
p('2'|'-'): 0.0
p('3'|'-'): 0.0
p('4'|'-'): 0.0
p('5'|'-'): 0.0
p('6'|'-'): 0.0010940919037199124
p('7'|'-'): 0.0
p('8'|'-'): 0.0005470459518599562
p('9'|'-'): 0.0
p('a'|'-'): 0.026805251641137857
p('b'|'-'): 0.06345733041575492
p('c'|'-'): 0.09573304157549234
p('d'|'-'): 0.08315098468271334
p('e'|'-'): 0.04266958424507659
p('f'|'-'): 0.07056892778993436
p('g'|'-'): 0.019146608315098467
p('h'|'-'): 0.03938730853391685
p('i'|'-'): 0.048687089715536105
p('j'|'-'): 0.0005470459518599562
p('k'|'-'): 0.019693654266958426
p('l'|'-'): 0.06455142231947483
p('m'|'-'): 0.03665207877461707
p('n'|'-'): 0.03665207877461707
p('o'|'-'): 0.05087527352297593
p('p'|'-'): 0.028446389496717725
p('q'|'-'): 0.0005470459518599562
p('r'|'-'): 0.02899343544857768
p('s'|'-'): 0.09026258205689278
```

```
p('t'|'-'): 0.07221006564551423
p('u'|'-'): 0.016411378555798686
p('v'|'-'): 0.0032822757111597373
p('w'|'-'): 0.023522975929978117
p('x'|'-'): 0.0
p('y'|'-'): 0.0175054704595186
p('z'|'-'): 0.006564551422319475


Symbol: ,
Marginal Probability p(','): 0.011806647074074533
Transition Probabilities:
p('='|','): 0.0
p(' '|','): 0.9996774627066255
p('-'|','): 0.0
p(','|','): 0.0
p(';'|','): 0.0
p(':'|','): 0.0
p('!'|','): 0.0
p('?'|','): 0.0
p('/'|','): 0.0
p('.'|','): 2.6878107781212204e-05
p('''|','): 0.0
p('"'|','): 2.6878107781212204e-05
p('('|','): 0.0
p(')'|','): 0.0
p('['|','): 0.0
p(']'|','): 0.0
p('*'|','): 2.6878107781212204e-05
p('0'|','): 0.00024190297003090982
p('1'|','): 0.0
p('2'|','): 0.0
p('3'|','): 0.0
p('4'|','): 0.0
p('5'|','): 0.0
p('6'|','): 0.0
p('7'|','): 0.0
p('8'|','): 0.0
p('9'|','): 0.0
p('a'|','): 0.0
p('b'|','): 0.0
p('c'|','): 0.0
p('d'|','): 0.0
p('e'|','): 0.0
p('f'|','): 0.0
p('g'|','): 0.0
p('h'|','): 0.0
```

```
p('i'|','): 0.0
p('j'|','): 0.0
p('k'|','): 0.0
p('l'|','): 0.0
p('m'|','): 0.0
p('n'|','): 0.0
p('o'|','): 0.0
p('p'|','): 0.0
p('q'|','): 0.0
p('r'|','): 0.0
p('s'|','): 0.0
p('t'|','): 0.0
p('u'|','): 0.0
p('v'|','): 0.0
p('w'|','): 0.0
p('x'|','): 0.0
p('y'|','): 0.0
p('z'|','): 0.0


Symbol: ;
Marginal Probability p(';'): 0.0003630373404849151
Transition Probabilities:
p('='|';'): 0.0
p(' '|';'): 1.0
p('-'|';'): 0.0
p(','|';'): 0.0
p(';'|';'): 0.0
p(':'|';'): 0.0
p('!'|';'): 0.0
p('?'|';'): 0.0
p('/'|';'): 0.0
p('.'|';'): 0.0
p('''|';'): 0.0
p('"'|';'): 0.0
p('('|';'): 0.0
p(')'|';'): 0.0
p('['|';'): 0.0
p(']'|';'): 0.0
p('*'|';'): 0.0
p('0'|';'): 0.0
p('1'|';'): 0.0
p('2'|';'): 0.0
p('3'|';'): 0.0
p('4'|';'): 0.0
p('5'|';'): 0.0
p('6'|';'): 0.0
```

```
p('7'|';'): 0.0
p('8'|';'): 0.0
p('9'|';'): 0.0
p('a'|';'): 0.0
p('b'|';'): 0.0
p('c'|';'): 0.0
p('d'|';'): 0.0
p('e'|';'): 0.0
p('f'|';'): 0.0
p('g'|';'): 0.0
p('h'|';'): 0.0
p('i'|';'): 0.0
p('j'|';'): 0.0
p('k'|';'): 0.0
p('l'|';'): 0.0
p('m'|';'): 0.0
p('n'|';'): 0.0
p('o'|';'): 0.0
p('p'|';'): 0.0
p('q'|';'): 0.0
p('r'|';'): 0.0
p('s'|';'): 0.0
p('t'|';'): 0.0
p('u'|';'): 0.0
p('v'|';'): 0.0
p('w'|';'): 0.0
p('x'|';'): 0.0
p('y'|';'): 0.0
p('z'|';'): 0.0


Symbol: :
Marginal Probability p(':'): 0.00031956171491985097
Transition Probabilities:
p('='|':'): 0.0
p(' '|':'): 0.9990069513406157
p('-'|':'): 0.0
p(','|':'): 0.0
p(';'|':'): 0.0
p(':'|':'): 0.0
p('!'|':'): 0.0
p('?'|':'): 0.0
p('/'|':'): 0.0
p('.'|':'): 0.0
p('''|':'): 0.0
p('"'|':'): 0.0
p('('|':'): 0.0
```

```
p(')'|':'): 0.0
p('['|':'): 0.0
p(']'|':'): 0.0
p('*'|':'): 0.0009930486593843098
p('0'|':'): 0.0
p('1'|':'): 0.0
p('2'|':'): 0.0
p('3'|':'): 0.0
p('4'|':'): 0.0
p('5'|':'): 0.0
p('6'|':'): 0.0
p('7'|':'): 0.0
p('8'|':'): 0.0
p('9'|':'): 0.0
p('a'|':'): 0.0
p('b'|':'): 0.0
p('c'|':'): 0.0
p('d'|':'): 0.0
p('e'|':'): 0.0
p('f'|':'): 0.0
p('g'|':'): 0.0
p('h'|':'): 0.0
p('i'|':'): 0.0
p('j'|':'): 0.0
p('k'|':'): 0.0
p('l'|':'): 0.0
p('m'|':'): 0.0
p('n'|':'): 0.0
p('o'|':'): 0.0
p('p'|':'): 0.0
p('q'|':'): 0.0
p('r'|':'): 0.0
p('s'|':'): 0.0
p('t'|':'): 0.0
p('u'|':'): 0.0
p('v'|':'): 0.0
p('w'|':'): 0.0
p('x'|':'): 0.0
p('y'|':'): 0.0
p('z'|':'): 0.0


Symbol: !
Marginal Probability p('!'): 0.0005791461069798689
Transition Probabilities:
p('='|'!'): 0.0
p(' '|'!'): 0.9019178082191781
```

```
p('-'|'!'): 0.0
p(','|'!'): 0.0
p(';'|'!'): 0.0
p(':'|'!'): 0.0
p('!'|'!'): 0.0
p('?'|'!'): 0.0
p('/'|'!'): 0.0
p('.'|'!'): 0.0958904109589041
p('''|'!'): 0.0
p('"'|'!'): 0.0
p('('|'!'): 0.0
p(')'|'!'): 0.0016438356164383563
p('['|'!'): 0.0
p(']'|'!'): 0.0
p('*'|'!'): 0.000547945205479452
p('0'|'!'): 0.0
p('1'|'!'): 0.0
p('2'|'!'): 0.0
p('3'|'!'): 0.0
p('4'|'!'): 0.0
p('5'|'!'): 0.0
p('6'|'!'): 0.0
p('7'|'!'): 0.0
p('8'|'!'): 0.0
p('9'|'!'): 0.0
p('a'|'!'): 0.0
p('b'|'!'): 0.0
p('c'|'!'): 0.0
p('d'|'!'): 0.0
p('e'|'!'): 0.0
p('f'|'!'): 0.0
p('g'|'!'): 0.0
p('h'|'!'): 0.0
p('i'|'!'): 0.0
p('j'|'!'): 0.0
p('k'|'!'): 0.0
p('l'|'!'): 0.0
p('m'|'!'): 0.0
p('n'|'!'): 0.0
p('o'|'!'): 0.0
p('p'|'!'): 0.0
p('q'|'!'): 0.0
p('r'|'!'): 0.0
p('s'|'!'): 0.0
p('t'|'!'): 0.0
p('u'|'!'): 0.0
p('v'|'!'): 0.0
```

```
p('w'|'!'): 0.0
p('x'|'!'): 0.0
p('y'|'!'): 0.0
p('z'|'!'): 0.0


Symbol: ?
Marginal Probability p('?'): 0.0004515752932780019
Transition Probabilities:
p('='|'?'): 0.0
p(' '|'?'): 0.9163738580463809
p('-'|'?'): 0.0
p(','|'?'): 0.0
p(';'|'?'): 0.0
p(':'|'?'): 0.0
p('!'|'?'): 0.0
p('?'|'?'): 0.0
p('/'|'?'): 0.0
p('.'|'?'): 0.08222066057624737
p('''|'?'): 0.0
p('"'|'?'): 0.0
p('('|'?'): 0.0
p(')'|'?'): 0.0014054813773717498
p('['|'?'): 0.0
p(']'|'?'): 0.0
p('*'|'?'): 0.0
p('0'|'?'): 0.0
p('1'|'?'): 0.0
p('2'|'?'): 0.0
p('3'|'?'): 0.0
p('4'|'?'): 0.0
p('5'|'?'): 0.0
p('6'|'?'): 0.0
p('7'|'?'): 0.0
p('8'|'?'): 0.0
p('9'|'?'): 0.0
p('a'|'?'): 0.0
p('b'|'?'): 0.0
p('c'|'?'): 0.0
p('d'|'?'): 0.0
p('e'|'?'): 0.0
p('f'|'?'): 0.0
p('g'|'?'): 0.0
p('h'|'?'): 0.0
p('i'|'?'): 0.0
p('j'|'?'): 0.0
p('k'|'?'): 0.0
```

```
p('l'|'?'): 0.0
p('m'|'?'): 0.0
p('n'|'?'): 0.0
p('o'|'?'): 0.0
p('p'|'?'): 0.0
p('q'|'?'): 0.0
p('r'|'?'): 0.0
p('s'|'?'): 0.0
p('t'|'?'): 0.0
p('u'|'?'): 0.0
p('v'|'?'): 0.0
p('w'|'?'): 0.0
p('x'|'?'): 0.0
p('y'|'?'): 0.0
p('z'|'?'): 0.0


Symbol: /
Marginal Probability p('/'): 2.856062993325381e-06
Transition Probabilities:
p('='|'/'): 0.0
p(' '|'/'): 0.0
p('-'|'/'): 0.0
p(','|'/'): 0.0
p(';'|'/'): 0.0
p(':'|'/'): 0.0
p('!'|'/'): 0.0
p('?'|'/'): 0.0
p('/'|'/'): 0.0
p('.'|'/'): 0.0
p('''|'/'): 0.0
p('"'|'/'): 0.0
p('('|'/'): 0.0
p(')'|'/'): 0.0
p('['|'/'): 0.0
p(']'|'/'): 0.0
p('*'|'/'): 0.0
p('0'|'/'): 0.0
p('1'|'/'): 0.0
p('2'|'/'): 0.0
p('3'|'/'): 0.0
p('4'|'/'): 0.1111111111111111
p('5'|'/'): 0.0
p('6'|'/'): 0.0
p('7'|'/'): 0.0
p('8'|'/'): 0.0
p('9'|'/'): 0.0
```

```
p('a'|'/'): 0.0
p('b'|'/'): 0.0
p('c'|'/'): 0.2222222222222222
p('d'|'/'): 0.2222222222222222
p('e'|'/'): 0.0
p('f'|'/'): 0.0
p('g'|'/'): 0.0
p('h'|'/'): 0.1111111111111111
p('i'|'/'): 0.0
p('j'|'/'): 0.0
p('k'|'/'): 0.0
p('l'|'/'): 0.1111111111111111
p('m'|'/'): 0.0
p('n'|'/'): 0.0
p('o'|'/'): 0.0
p('p'|'/'): 0.0
p('q'|'/'): 0.0
p('r'|'/'): 0.0
p('s'|'/'): 0.0
p('t'|'/'): 0.1111111111111111
p('u'|'/'): 0.0
p('v'|'/'): 0.0
p('w'|'/'): 0.0
p('x'|'/'): 0.0
p('y'|'/'): 0.1111111111111111
p('z'|'/'): 0.0


Symbol: .
Marginal Probability p('.'): 0.009049911604850356
Transition Probabilities:
p('='|'.'): 0.0
p(' '|'.'): 0.8526193982747738
p('-'|'.'): 0.0
p(','|'.'): 0.0002805245809664072
p(';'|'.'): 0.0
p(':'|'.'): 0.0
p('!'|'.'): 0.0003857212988288099
p('?'|'.'): 0.0005610491619328144
p('/'|'.'): 0.0
p('.'|'.'): 0.13994670032961637
p('''|'.'): 0.0
p('"'|'.'): 3.50655726208009e-05
p('('|'.'): 0.0
p(')'|'.'): 0.002454590083456063
p('['|'.'): 0.0
p(']'|'.'): 0.0
```

```
p('*'|'.'): 3.50655726208009e-05
p('0'|'.'): 0.0
p('1'|'.'): 0.00021039343572480537
p('2'|'.'): 7.01311452416018e-05
p('3'|'.'): 0.00021039343572480537
p('4'|'.'): 7.01311452416018e-05
p('5'|'.'): 7.01311452416018e-05
p('6'|'.'): 7.01311452416018e-05
p('7'|'.'): 0.00010519671786240269
p('8'|'.'): 0.0001402622904832036
p('9'|'.'): 0.00010519671786240269
p('a'|'.'): 0.0001402622904832036
p('b'|'.'): 3.50655726208009e-05
p('c'|'.'): 7.01311452416018e-05
p('d'|'.'): 3.50655726208009e-05
p('e'|'.'): 0.0007714425976576198
p('f'|'.'): 0.0003857212988288099
p('g'|'.'): 0.0003155901535872081
p('h'|'.'): 0.0
p('i'|'.'): 0.0
p('j'|'.'): 0.0
p('k'|'.'): 0.0
p('l'|'.'): 3.50655726208009e-05
p('m'|'.'): 0.0
p('n'|'.'): 0.00010519671786240269
p('o'|'.'): 0.000350655726208009
p('p'|'.'): 0.0
p('q'|'.'): 0.0
p('r'|'.'): 0.0
p('s'|'.'): 0.0003155901535872081
p('t'|'.'): 3.50655726208009e-05
p('u'|'.'): 3.50655726208009e-05
p('v'|'.'): 0.0
p('w'|'.'): 0.0
p('x'|'.'): 0.0
p('y'|'.'): 0.0
p('z'|'.'): 0.0


Symbol: '
Marginal Probability p('''): 2.221382328141963e-06
Transition Probabilities:
p('='|'''): 0.0
p(' '|'''): 0.0
p('-'|'''): 0.0
p(','|'''): 0.14285714285714285
p(';'|'''): 0.0
```

```
p(':'|'''): 0.0
p('!'|'''): 0.0
p('?'|'''): 0.0
p('/'|'''): 0.0
p('.'|'''): 0.0
p('''|'''): 0.0
p('"'|'''): 0.0
p('('|'''): 0.0
p(')'|'''): 0.0
p('['|'''): 0.0
p(']'|'''): 0.0
p('*'|'''): 0.0
p('0'|'''): 0.0
p('1'|'''): 0.0
p('2'|'''): 0.0
p('3'|'''): 0.0
p('4'|'''): 0.0
p('5'|'''): 0.0
p('6'|'''): 0.0
p('7'|'''): 0.0
p('8'|'''): 0.0
p('9'|'''): 0.0
p('a'|'''): 0.14285714285714285
p('b'|'''): 0.0
p('c'|'''): 0.0
p('d'|'''): 0.0
p('e'|'''): 0.0
p('f'|'''): 0.0
p('g'|'''): 0.0
p('h'|'''): 0.0
p('i'|'''): 0.0
p('j'|'''): 0.0
p('k'|'''): 0.0
p('l'|'''): 0.0
p('m'|'''): 0.0
p('n'|'''): 0.0
p('o'|'''): 0.0
p('p'|'''): 0.0
p('q'|'''): 0.0
p('r'|'''): 0.0
p('s'|'''): 0.7142857142857143
p('t'|'''): 0.0
p('u'|'''): 0.0
p('v'|'''): 0.0
p('w'|'''): 0.0
p('x'|'''): 0.0
p('y'|'''): 0.0
```

```
p('z'|'''): 0.0


Symbol: "
Marginal Probability p('"'): 6.9814873170175976e-06
Transition Probabilities:
p('='|'"'): 0.0
p(' '|'"'): 0.45454545454545453
p('-'|'"'): 0.0
p(','|'"'): 0.0
p(';'|'"'): 0.0
p(':'|'"'): 0.0
p('!'|'"'): 0.0
p('?'|'"'): 0.0
p('/'|'"'): 0.0
p('.'|'"'): 0.0
p('''|'"'): 0.0
p('"'|'"'): 0.0
p('('|'"'): 0.0
p(')'|'"'): 0.045454545454545456
p('['|'"'): 0.0
p(']'|'"'): 0.0
p('*'|'"'): 0.0
p('0'|'"'): 0.0
p('1'|'"'): 0.0
p('2'|'"'): 0.0
p('3'|'"'): 0.0
p('4'|'"'): 0.0
p('5'|'"'): 0.0
p('6'|'"'): 0.0
p('7'|'"'): 0.0
p('8'|'"'): 0.0
p('9'|'"'): 0.0
p('a'|'"'): 0.0
p('b'|'"'): 0.0
p('c'|'"'): 0.0
p('d'|'"'): 0.045454545454545456
p('e'|'"'): 0.0
p('f'|'"'): 0.0
p('g'|'"'): 0.0
p('h'|'"'): 0.0
p('i'|'"'): 0.045454545454545456
p('j'|'"'): 0.0
p('k'|'"'): 0.0
p('l'|'"'): 0.0
p('m'|'"'): 0.0
p('n'|'"'): 0.0
```

```
p('o'|'"'): 0.0
p('p'|'"'): 0.3181818181818182
p('q'|'"'): 0.0
p('r'|'"'): 0.045454545454545456
p('s'|'"'): 0.0
p('t'|'"'): 0.045454545454545456
p('u'|'"'): 0.0
p('v'|'"'): 0.0
p('w'|'"'): 0.0
p('x'|'"'): 0.0
p('y'|'"'): 0.0
p('z'|'"'): 0.0


Symbol: (
Marginal Probability p('('): 0.00020944461951052793
Transition Probabilities:
p('='|'('): 0.0
p(' '|'('): 0.0
p('-'|'('): 0.0
p(','|'('): 0.0
p(';'|'('): 0.0
p(':'|'('): 0.0
p('!'|'('): 0.0
p('?'|'('): 0.0
p('/'|'('): 0.0
p('.'|'('): 0.0
p('''|'('): 0.0
p('"'|'('): 0.0015151515151515152
p('('|'('): 0.0
p(')'|'('): 0.0
p('['|'('): 0.0
p(']'|'('): 0.0
p('*'|'('): 0.0
p('0'|'('): 0.0
p('1'|'('): 0.025757575757575757
p('2'|'('): 0.05909090909090909
p('3'|'('): 0.022727272727272728
p('4'|'('): 0.004545454545454545
p('5'|'('): 0.0015151515151515152
p('6'|'('): 0.0
p('7'|'('): 0.0
p('8'|'('): 0.0015151515151515152
p('9'|'('): 0.0
p('a'|'('): 0.12424242424242424
p('b'|'('): 0.024242424242424242
p('c'|'('): 0.007575757575757576
```

```
p('d'|'('): 0.010606060606060607
p('e'|'('): 0.010606060606060607
p('f'|'('): 0.021212121212121213
p('g'|'('): 0.006060606060606061
p('h'|'('): 0.11515151515151516
p('i'|'('): 0.06515151515151515
p('j'|'('): 0.0015151515151515152
p('k'|'('): 0.007575757575757576
p('l'|'('): 0.00909090909090909
p('m'|'('): 0.01818181818181818
p('n'|'('): 0.025757575757575757
p('o'|'('): 0.05
p('p'|'('): 0.03484848484848485
p('q'|'('): 0.0015151515151515152
p('r'|'('): 0.006060606060606061
p('s'|'('): 0.06666666666666667
p('t'|'('): 0.15151515151515152
p('u'|'('): 0.0030303030303030303
p('v'|'('): 0.004545454545454545
p('w'|'('): 0.11666666666666667
p('x'|'('): 0.0
p('y'|'('): 0.0015151515151515152
p('z'|'('): 0.0


Symbol: )
Marginal Probability p(')'): 0.0002078579178475694
Transition Probabilities:
p('='|')'): 0.0
p(' '|')'): 0.6748091603053435
p('-'|')'): 0.0
p(','|')'): 0.2595419847328244
p(';'|')'): 0.007633587786259542
p(':'|')'): 0.0015267175572519084
p('!'|')'): 0.0
p('?'|')'): 0.0
p('/'|')'): 0.0
p('.'|')'): 0.0549618320610687
p('''|')'): 0.0
p('"'|')'): 0.0
p('('|')'): 0.0015267175572519084
p(')'|')'): 0.0
p('['|')'): 0.0
p(']'|')'): 0.0
p('*'|')'): 0.0
p('0'|')'): 0.0
p('1'|')'): 0.0
```

```
p('2'|')'): 0.0
p('3'|')'): 0.0
p('4'|')'): 0.0
p('5'|')'): 0.0
p('6'|')'): 0.0
p('7'|')'): 0.0
p('8'|')'): 0.0
p('9'|')'): 0.0
p('a'|')'): 0.0
p('b'|')'): 0.0
p('c'|')'): 0.0
p('d'|')'): 0.0
p('e'|')'): 0.0
p('f'|')'): 0.0
p('g'|')'): 0.0
p('h'|')'): 0.0
p('i'|')'): 0.0
p('j'|')'): 0.0
p('k'|')'): 0.0
p('l'|')'): 0.0
p('m'|')'): 0.0
p('n'|')'): 0.0
p('o'|')'): 0.0
p('p'|')'): 0.0
p('q'|')'): 0.0
p('r'|')'): 0.0
p('s'|')'): 0.0
p('t'|')'): 0.0
p('u'|')'): 0.0
p('v'|')'): 0.0
p('w'|')'): 0.0
p('x'|')'): 0.0
p('y'|')'): 0.0
p('z'|')'): 0.0


Symbol: [
Marginal Probability p('['): 6.34680665183418e-07
Transition Probabilities:
p('='|'['): 0.0
p(' '|'['): 0.0
p('-'|'['): 0.0
p(','|'['): 0.0
p(';'|'['): 0.0
p(':'|'['): 0.0
p('!'|'['): 0.0
p('?'|'['): 0.0
```

```
p('/'|'['): 0.0
p('.'|'['): 0.0
p('''|'['): 0.0
p('"'|'['): 0.0
p('('|'['): 0.0
p(')'|'['): 0.0
p('['|'['): 0.0
p(']'|'['): 0.0
p('*'|'['): 0.0
p('0'|'['): 0.0
p('1'|'['): 0.0
p('2'|'['): 0.0
p('3'|'['): 0.0
p('4'|'['): 0.0
p('5'|'['): 0.0
p('6'|'['): 0.0
p('7'|'['): 0.0
p('8'|'['): 0.0
p('9'|'['): 0.0
p('a'|'['): 0.0
p('b'|'['): 0.0
p('c'|'['): 0.0
p('d'|'['): 0.0
p('e'|'['): 0.5
p('f'|'['): 0.0
p('g'|'['): 0.0
p('h'|'['): 0.0
p('i'|'['): 0.0
p('j'|'['): 0.0
p('k'|'['): 0.0
p('l'|'['): 0.0
p('m'|'['): 0.5
p('n'|'['): 0.0
p('o'|'['): 0.0
p('p'|'['): 0.0
p('q'|'['): 0.0
p('r'|'['): 0.0
p('s'|'['): 0.0
p('t'|'['): 0.0
p('u'|'['): 0.0
p('v'|'['): 0.0
p('w'|'['): 0.0
p('x'|'['): 0.0
p('y'|'['): 0.0
p('z'|'['): 0.0

```

```
Symbol: ]
Marginal Probability p(']'): 6.34680665183418e-07
Transition Probabilities:
p('='|']'): 0.0
p(' '|']'): 1.0
p('-'|']'): 0.0
p(','|']'): 0.0
p(';'|']'): 0.0
p(':'|']'): 0.0
p('!'|']'): 0.0
p('?'|']'): 0.0
p('/'|']'): 0.0
p('.'|']'): 0.0
p('''|']'): 0.0
p('"'|']'): 0.0
p('('|']'): 0.0
p(')'|']'): 0.0
p('['|']'): 0.0
p(']'|']'): 0.0
p('*'|']'): 0.0
p('0'|']'): 0.0
p('1'|']'): 0.0
p('2'|']'): 0.0
p('3'|']'): 0.0
p('4'|']'): 0.0
p('5'|']'): 0.0
p('6'|']'): 0.0
p('7'|']'): 0.0
p('8'|']'): 0.0
p('9'|']'): 0.0
p('a'|']'): 0.0
p('b'|']'): 0.0
p('c'|']'): 0.0
p('d'|']'): 0.0
p('e'|']'): 0.0
p('f'|']'): 0.0
p('g'|']'): 0.0
p('h'|']'): 0.0
p('i'|']'): 0.0
p('j'|']'): 0.0
p('k'|']'): 0.0
p('l'|']'): 0.0
p('m'|']'): 0.0
p('n'|']'): 0.0
p('o'|']'): 0.0
p('p'|']'): 0.0
p('q'|']'): 0.0
```

```
p('r'|']'): 0.0
p('s'|']'): 0.0
p('t'|']'): 0.0
p('u'|']'): 0.0
p('v'|']'): 0.0
p('w'|']'): 0.0
p('x'|']'): 0.0
p('y'|']'): 0.0
p('z'|']'): 0.0


Symbol: *
Marginal Probability p('*'): 9.075933512122877e-05
Transition Probabilities:
p('='|'*'): 0.0
p(' '|'*'): 0.9020979020979021
p('-'|'*'): 0.0
p(','|'*'): 0.0
p(';'|'*'): 0.0
p(':'|'*'): 0.0
p('!'|'*'): 0.0
p('?'|'*'): 0.0
p('/'|'*'): 0.0
p('.'|'*'): 0.0
p('''|'*'): 0.0
p('"'|'*'): 0.0
p('('|'*'): 0.055944055944055944
p(')'|'*'): 0.0
p('['|'*'): 0.0
p(']'|'*'): 0.0
p('*'|'*'): 0.027972027972027972
p('0'|'*'): 0.0
p('1'|'*'): 0.0
p('2'|'*'): 0.0
p('3'|'*'): 0.0
p('4'|'*'): 0.0
p('5'|'*'): 0.0
p('6'|'*'): 0.0
p('7'|'*'): 0.0
p('8'|'*'): 0.0
p('9'|'*'): 0.0
p('a'|'*'): 0.0
p('b'|'*'): 0.0
p('c'|'*'): 0.0034965034965034965
p('d'|'*'): 0.0034965034965034965
p('e'|'*'): 0.0
p('f'|'*'): 0.0
```

```
p('g'|'*'): 0.0
p('h'|'*'): 0.0
p('i'|'*'): 0.0
p('j'|'*'): 0.0
p('k'|'*'): 0.0034965034965034965
p('l'|'*'): 0.0
p('m'|'*'): 0.0
p('n'|'*'): 0.0
p('o'|'*'): 0.0
p('p'|'*'): 0.0034965034965034965
p('q'|'*'): 0.0
p('r'|'*'): 0.0
p('s'|'*'): 0.0
p('t'|'*'): 0.0
p('u'|'*'): 0.0
p('v'|'*'): 0.0
p('w'|'*'): 0.0
p('x'|'*'): 0.0
p('y'|'*'): 0.0
p('z'|'*'): 0.0


Symbol: 0
Marginal Probability p('0'): 5.331317587540711e-05
Transition Probabilities:
p('='|'0'): 0.0
p(' '|'0'): 0.2619047619047619
p('-'|'0'): 0.0
p(','|'0'): 0.06547619047619048
p(';'|'0'): 0.0
p(':'|'0'): 0.0
p('!'|'0'): 0.0
p('?'|'0'): 0.0
p('/'|'0'): 0.0
p('.'|'0'): 0.005952380952380952
p('''|'0'): 0.0
p('"'|'0'): 0.0
p('('|'0'): 0.0
p(')'|'0'): 0.005952380952380952
p('['|'0'): 0.0
p(']'|'0'): 0.005952380952380952
p('*'|'0'): 0.0
p('0'|'0'): 0.16071428571428573
p('1'|'0'): 0.023809523809523808
p('2'|'0'): 0.017857142857142856
p('3'|'0'): 0.0
p('4'|'0'): 0.0
```

```
p('5'|'0'): 0.16071428571428573
p('6'|'0'): 0.07142857142857142
p('7'|'0'): 0.08928571428571429
p('8'|'0'): 0.02976190476190476
p('9'|'0'): 0.10119047619047619
p('a'|'0'): 0.0
p('b'|'0'): 0.0
p('c'|'0'): 0.0
p('d'|'0'): 0.0
p('e'|'0'): 0.0
p('f'|'0'): 0.0
p('g'|'0'): 0.0
p('h'|'0'): 0.0
p('i'|'0'): 0.0
p('j'|'0'): 0.0
p('k'|'0'): 0.0
p('l'|'0'): 0.0
p('m'|'0'): 0.0
p('n'|'0'): 0.0
p('o'|'0'): 0.0
p('p'|'0'): 0.0
p('q'|'0'): 0.0
p('r'|'0'): 0.0
p('s'|'0'): 0.0
p('t'|'0'): 0.0
p('u'|'0'): 0.0
p('v'|'0'): 0.0
p('w'|'0'): 0.0
p('x'|'0'): 0.0
p('y'|'0'): 0.0
p('z'|'0'): 0.0


Symbol: 1
Marginal Probability p('1'): 0.0001237627297107665
Transition Probabilities:
p('='|'1'): 0.0
p(' '|'1'): 0.046153846153846156
p('-'|'1'): 0.0
p(','|'1'): 0.010256410256410256
p(';'|'1'): 0.0
p(':'|'1'): 0.0
p('!'|'1'): 0.0
p('?'|'1'): 0.0
p('/'|'1'): 0.0
p('.'|'1'): 0.12051282051282051
p('''|'1'): 0.0
```

```
p('"'|'1'): 0.0
p('('|'1'): 0.002564102564102564
p(')'|'1'): 0.046153846153846156
p('['|'1'): 0.0
p(']'|'1'): 0.0
p('*'|'1'): 0.0
p('0'|'1'): 0.038461538461538464
p('1'|'1'): 0.041025641025641026
p('2'|'1'): 0.15897435897435896
p('3'|'1'): 0.05128205128205128
p('4'|'1'): 0.005128205128205128
p('5'|'1'): 0.023076923076923078
p('6'|'1'): 0.007692307692307693
p('7'|'1'): 0.020512820512820513
p('8'|'1'): 0.4256410256410256
p('9'|'1'): 0.0
p('a'|'1'): 0.0
p('b'|'1'): 0.0
p('c'|'1'): 0.0
p('d'|'1'): 0.0
p('e'|'1'): 0.0
p('f'|'1'): 0.0
p('g'|'1'): 0.0
p('h'|'1'): 0.0
p('i'|'1'): 0.0
p('j'|'1'): 0.0
p('k'|'1'): 0.0
p('l'|'1'): 0.0
p('m'|'1'): 0.0
p('n'|'1'): 0.0
p('o'|'1'): 0.0
p('p'|'1'): 0.0
p('q'|'1'): 0.0
p('r'|'1'): 0.0
p('s'|'1'): 0.002564102564102564
p('t'|'1'): 0.0
p('u'|'1'): 0.0
p('v'|'1'): 0.0
p('w'|'1'): 0.0
p('x'|'1'): 0.0
p('y'|'1'): 0.0
p('z'|'1'): 0.0


Symbol: 2
Marginal Probability p('2'): 4.474498689543097e-05
Transition Probabilities:
```

```
p('='|'2'): 0.0
p(' '|'2'): 0.2978723404255319
p('-'|'2'): 0.0
p(','|'2'): 0.09219858156028368
p(';'|'2'): 0.0
p(':'|'2'): 0.0
p('!'|'2'): 0.0
p('?'|'2'): 0.0070921985815602835
p('/'|'2'): 0.0
p('.'|'2'): 0.09219858156028368
p('''|'2'): 0.0
p('"'|'2'): 0.0
p('('|'2'): 0.0
p(')'|'2'): 0.2765957446808511
p('['|'2'): 0.0
p(']'|'2'): 0.0070921985815602835
p('*'|'2'): 0.0
p('0'|'2'): 0.0851063829787234
p('1'|'2'): 0.014184397163120567
p('2'|'2'): 0.03546099290780142
p('3'|'2'): 0.014184397163120567
p('4'|'2'): 0.02127659574468085
p('5'|'2'): 0.02127659574468085
p('6'|'2'): 0.014184397163120567
p('7'|'2'): 0.014184397163120567
p('8'|'2'): 0.0
p('9'|'2'): 0.0
p('a'|'2'): 0.0
p('b'|'2'): 0.0
p('c'|'2'): 0.0
p('d'|'2'): 0.0
p('e'|'2'): 0.0
p('f'|'2'): 0.0
p('g'|'2'): 0.0
p('h'|'2'): 0.0
p('i'|'2'): 0.0
p('j'|'2'): 0.0
p('k'|'2'): 0.0
p('l'|'2'): 0.0
p('m'|'2'): 0.0
p('n'|'2'): 0.0070921985815602835
p('o'|'2'): 0.0
p('p'|'2'): 0.0
p('q'|'2'): 0.0
p('r'|'2'): 0.0
p('s'|'2'): 0.0
p('t'|'2'): 0.0
```

```
p('u'|'2'): 0.0
p('v'|'2'): 0.0
p('w'|'2'): 0.0
p('x'|'2'): 0.0
p('y'|'2'): 0.0
p('z'|'2'): 0.0


Symbol: 3
Marginal Probability p('3'): 1.8405739290319122e-05
Transition Probabilities:
p('='|'3'): 0.0
p(' '|'3'): 0.22413793103448276
p('-'|'3'): 0.0
p(','|'3'): 0.10344827586206896
p(';'|'3'): 0.0
p(':'|'3'): 0.0
p('!'|'3'): 0.0
p('?'|'3'): 0.0
p('/'|'3'): 0.0
p('.'|'3'): 0.1206896551724138
p('''|'3'): 0.0
p('"'|'3'): 0.0
p('('|'3'): 0.0
p(')'|'3'): 0.27586206896551724
p('['|'3'): 0.0
p(']'|'3'): 0.0
p('*'|'3'): 0.0
p('0'|'3'): 0.1206896551724138
p('1'|'3'): 0.017241379310344827
p('2'|'3'): 0.0
p('3'|'3'): 0.0
p('4'|'3'): 0.0
p('5'|'3'): 0.0
p('6'|'3'): 0.0
p('7'|'3'): 0.0
p('8'|'3'): 0.0
p('9'|'3'): 0.0
p('a'|'3'): 0.0
p('b'|'3'): 0.0
p('c'|'3'): 0.0
p('d'|'3'): 0.0
p('e'|'3'): 0.0
p('f'|'3'): 0.0
p('g'|'3'): 0.0
p('h'|'3'): 0.0
p('i'|'3'): 0.0
```

```
p('j'|'3'): 0.0
p('k'|'3'): 0.0
p('l'|'3'): 0.0
p('m'|'3'): 0.0
p('n'|'3'): 0.0
p('o'|'3'): 0.0
p('p'|'3'): 0.0
p('q'|'3'): 0.0
p('r'|'3'): 0.08620689655172414
p('s'|'3'): 0.0
p('t'|'3'): 0.05172413793103448
p('u'|'3'): 0.0
p('v'|'3'): 0.0
p('w'|'3'): 0.0
p('x'|'3'): 0.0
p('y'|'3'): 0.0
p('z'|'3'): 0.0


Symbol: 4
Marginal Probability p('4'): 7.2988276496093065e-06
Transition Probabilities:
p('='|'4'): 0.0
p(' '|'4'): 0.17391304347826086
p('-'|'4'): 0.043478260869565216
p(','|'4'): 0.08695652173913043
p(';'|'4'): 0.0
p(':'|'4'): 0.0
p('!'|'4'): 0.0
p('?'|'4'): 0.0
p('/'|'4'): 0.0
p('.'|'4'): 0.2608695652173913
p('''|'4'): 0.0
p('"'|'4'): 0.0
p('('|'4'): 0.0
p(')'|'4'): 0.13043478260869565
p('['|'4'): 0.0
p(']'|'4'): 0.0
p('*'|'4'): 0.0
p('0'|'4'): 0.08695652173913043
p('1'|'4'): 0.08695652173913043
p('2'|'4'): 0.0
p('3'|'4'): 0.0
p('4'|'4'): 0.0
p('5'|'4'): 0.0
p('6'|'4'): 0.0
p('7'|'4'): 0.0
```

```
p('8'|'4'): 0.0
p('9'|'4'): 0.0
p('a'|'4'): 0.0
p('b'|'4'): 0.0
p('c'|'4'): 0.0
p('d'|'4'): 0.0
p('e'|'4'): 0.0
p('f'|'4'): 0.0
p('g'|'4'): 0.0
p('h'|'4'): 0.0
p('i'|'4'): 0.0
p('j'|'4'): 0.0
p('k'|'4'): 0.0
p('l'|'4'): 0.0
p('m'|'4'): 0.0
p('n'|'4'): 0.0
p('o'|'4'): 0.0
p('p'|'4'): 0.0
p('q'|'4'): 0.0
p('r'|'4'): 0.0
p('s'|'4'): 0.0
p('t'|'4'): 0.08695652173913043
p('u'|'4'): 0.0
p('v'|'4'): 0.0
p('w'|'4'): 0.0
p('x'|'4'): 0.043478260869565216
p('y'|'4'): 0.0
p('z'|'4'): 0.0


Symbol: 5
Marginal Probability p('5'): 1.586701662958545e-05
Transition Probabilities:
p('='|'5'): 0.0
p(' '|'5'): 0.22
p('-'|'5'): 0.02
p(','|'5'): 0.32
p(';'|'5'): 0.02
p(':'|'5'): 0.02
p('!'|'5'): 0.0
p('?'|'5'): 0.0
p('/'|'5'): 0.02
p('.'|'5'): 0.16
p('''|'5'): 0.0
p('"'|'5'): 0.0
p('('|'5'): 0.0
p(')'|'5'): 0.02
```

```
p('['|'5'): 0.0
p(']'|'5'): 0.0
p('*'|'5'): 0.0
p('0'|'5'): 0.12
p('1'|'5'): 0.0
p('2'|'5'): 0.0
p('3'|'5'): 0.0
p('4'|'5'): 0.02
p('5'|'5'): 0.0
p('6'|'5'): 0.0
p('7'|'5'): 0.0
p('8'|'5'): 0.0
p('9'|'5'): 0.02
p('a'|'5'): 0.0
p('b'|'5'): 0.0
p('c'|'5'): 0.0
p('d'|'5'): 0.0
p('e'|'5'): 0.0
p('f'|'5'): 0.0
p('g'|'5'): 0.0
p('h'|'5'): 0.0
p('i'|'5'): 0.0
p('j'|'5'): 0.0
p('k'|'5'): 0.0
p('l'|'5'): 0.0
p('m'|'5'): 0.0
p('n'|'5'): 0.0
p('o'|'5'): 0.0
p('p'|'5'): 0.0
p('q'|'5'): 0.0
p('r'|'5'): 0.0
p('s'|'5'): 0.0
p('t'|'5'): 0.02
p('u'|'5'): 0.0
p('v'|'5'): 0.0
p('w'|'5'): 0.0
p('x'|'5'): 0.0
p('y'|'5'): 0.02
p('z'|'5'): 0.0


Symbol: 6
Marginal Probability p('6'): 1.650169729476887e-05
Transition Probabilities:
p('='|'6'): 0.0
p(' '|'6'): 0.21153846153846154
p('-'|'6'): 0.038461538461538464
```

```
p(','|'6'): 0.15384615384615385
p(';'|'6'): 0.019230769230769232
p(':'|'6'): 0.0
p('!'|'6'): 0.0
p('?'|'6'): 0.019230769230769232
p('/'|'6'): 0.0
p('.'|'6'): 0.09615384615384616
p('''|'6'): 0.0
p('"'|'6'): 0.0
p('('|'6'): 0.0
p(')'|'6'): 0.0
p('['|'6'): 0.0
p(']'|'6'): 0.0
p('*'|'6'): 0.0
p('0'|'6'): 0.09615384615384616
p('1'|'6'): 0.0
p('2'|'6'): 0.038461538461538464
p('3'|'6'): 0.0
p('4'|'6'): 0.019230769230769232
p('5'|'6'): 0.0
p('6'|'6'): 0.23076923076923078
p('7'|'6'): 0.019230769230769232
p('8'|'6'): 0.0
p('9'|'6'): 0.0
p('a'|'6'): 0.0
p('b'|'6'): 0.0
p('c'|'6'): 0.0
p('d'|'6'): 0.0
p('e'|'6'): 0.0
p('f'|'6'): 0.0
p('g'|'6'): 0.0
p('h'|'6'): 0.0
p('i'|'6'): 0.0
p('j'|'6'): 0.0
p('k'|'6'): 0.0
p('l'|'6'): 0.0
p('m'|'6'): 0.0
p('n'|'6'): 0.0
p('o'|'6'): 0.0
p('p'|'6'): 0.0
p('q'|'6'): 0.0
p('r'|'6'): 0.0
p('s'|'6'): 0.0
p('t'|'6'): 0.057692307692307696
p('u'|'6'): 0.0
p('v'|'6'): 0.0
p('w'|'6'): 0.0
```

```
p('x'|'6'): 0.0
p('y'|'6'): 0.0
p('z'|'6'): 0.0


Symbol: 7
Marginal Probability p('7'): 1.2058932638484941e-05
Transition Probabilities:
p('='|'7'): 0.0
p(' '|'7'): 0.3684210526315789
p('-'|'7'): 0.0
p(','|'7'): 0.18421052631578946
p(';'|'7'): 0.0
p(':'|'7'): 0.0
p('!'|'7'): 0.02631578947368421
p('?'|'7'): 0.0
p('/'|'7'): 0.0
p('.'|'7'): 0.10526315789473684
p('''|'7'): 0.0
p('"'|'7'): 0.0
p('('|'7'): 0.0
p(')'|'7'): 0.0
p('['|'7'): 0.0
p(']'|'7'): 0.0
p('*'|'7'): 0.0
p('0'|'7'): 0.07894736842105263
p('1'|'7'): 0.02631578947368421
p('2'|'7'): 0.0
p('3'|'7'): 0.0
p('4'|'7'): 0.0
p('5'|'7'): 0.0
p('6'|'7'): 0.0
p('7'|'7'): 0.0
p('8'|'7'): 0.10526315789473684
p('9'|'7'): 0.02631578947368421
p('a'|'7'): 0.0
p('b'|'7'): 0.0
p('c'|'7'): 0.0
p('d'|'7'): 0.0
p('e'|'7'): 0.0
p('f'|'7'): 0.0
p('g'|'7'): 0.0
p('h'|'7'): 0.0
p('i'|'7'): 0.0
p('j'|'7'): 0.0
p('k'|'7'): 0.0
p('l'|'7'): 0.0
```

```
p('m'|'7'): 0.0
p('n'|'7'): 0.0
p('o'|'7'): 0.0
p('p'|'7'): 0.0
p('q'|'7'): 0.0
p('r'|'7'): 0.0
p('s'|'7'): 0.0
p('t'|'7'): 0.07894736842105263
p('u'|'7'): 0.0
p('v'|'7'): 0.0
p('w'|'7'): 0.0
p('x'|'7'): 0.0
p('y'|'7'): 0.0
p('z'|'7'): 0.0


Symbol: 8
Marginal Probability p('8'): 6.0929343857608124e-05
Transition Probabilities:
p('='|'8'): 0.0
p(' '|'8'): 0.046875
p('-'|'8'): 0.0
p(','|'8'): 0.010416666666666666
p(';'|'8'): 0.0
p(':'|'8'): 0.0
p('!'|'8'): 0.0
p('?'|'8'): 0.0
p('/'|'8'): 0.0
p('.'|'8'): 0.020833333333333332
p('''|'8'): 0.0
p('"'|'8'): 0.0
p('('|'8'): 0.0
p(')'|'8'): 0.0
p('['|'8'): 0.0
p(']'|'8'): 0.0
p('*'|'8'): 0.0
p('0'|'8'): 0.4114583333333333
p('1'|'8'): 0.4427083333333333
p('2'|'8'): 0.020833333333333332
p('3'|'8'): 0.0
p('4'|'8'): 0.005208333333333333
p('5'|'8'): 0.0
p('6'|'8'): 0.005208333333333333
p('7'|'8'): 0.005208333333333333
p('8'|'8'): 0.005208333333333333
p('9'|'8'): 0.015625
p('a'|'8'): 0.0
```

```
p('b'|'8'): 0.0
p('c'|'8'): 0.0
p('d'|'8'): 0.0
p('e'|'8'): 0.0
p('f'|'8'): 0.0
p('g'|'8'): 0.0
p('h'|'8'): 0.0
p('i'|'8'): 0.0
p('j'|'8'): 0.0
p('k'|'8'): 0.0
p('l'|'8'): 0.0
p('m'|'8'): 0.0
p('n'|'8'): 0.0
p('o'|'8'): 0.0
p('p'|'8'): 0.0
p('q'|'8'): 0.0
p('r'|'8'): 0.0
p('s'|'8'): 0.0
p('t'|'8'): 0.010416666666666666
p('u'|'8'): 0.0
p('v'|'8'): 0.0
p('w'|'8'): 0.0
p('x'|'8'): 0.0
p('y'|'8'): 0.0
p('z'|'8'): 0.0


Symbol: 9
Marginal Probability p('9'): 9.837550310342979e-06
Transition Probabilities:
p('='|'9'): 0.0
p(' '|'9'): 0.45161290322580644
p('-'|'9'): 0.0
p(','|'9'): 0.16129032258064516
p(';'|'9'): 0.0
p(':'|'9'): 0.03225806451612903
p('!'|'9'): 0.0
p('?'|'9'): 0.0
p('/'|'9'): 0.0
p('.'|'9'): 0.16129032258064516
p('''|'9'): 0.0
p('"'|'9'): 0.0
p('('|'9'): 0.0
p(')'|'9'): 0.0
p('['|'9'): 0.0
p(']'|'9'): 0.0
p('*'|'9'): 0.0
```

```
p('0'|'9'): 0.0967741935483871
p('1'|'9'): 0.0
p('2'|'9'): 0.0
p('3'|'9'): 0.0
p('4'|'9'): 0.0
p('5'|'9'): 0.0
p('6'|'9'): 0.03225806451612903
p('7'|'9'): 0.03225806451612903
p('8'|'9'): 0.0
p('9'|'9'): 0.0
p('a'|'9'): 0.0
p('b'|'9'): 0.0
p('c'|'9'): 0.0
p('d'|'9'): 0.0
p('e'|'9'): 0.0
p('f'|'9'): 0.0
p('g'|'9'): 0.0
p('h'|'9'): 0.0
p('i'|'9'): 0.0
p('j'|'9'): 0.0
p('k'|'9'): 0.0
p('l'|'9'): 0.0
p('m'|'9'): 0.0
p('n'|'9'): 0.0
p('o'|'9'): 0.0
p('p'|'9'): 0.0
p('q'|'9'): 0.0
p('r'|'9'): 0.0
p('s'|'9'): 0.0
p('t'|'9'): 0.03225806451612903
p('u'|'9'): 0.0
p('v'|'9'): 0.0
p('w'|'9'): 0.0
p('x'|'9'): 0.0
p('y'|'9'): 0.0
p('z'|'9'): 0.0


Symbol: a
Marginal Probability p('a'): 0.06418525566999905
Transition Probabilities:
p('='|'a'): 0.0
p(' '|'a'): 0.06588549391871848
p('-'|'a'): 0.000182932858696727
p(','|'a'): 0.004677148225056858
p(';'|'a'): 9.393849500642738e-05
p(':'|'a'): 2.966478789676654e-05
```

```
p('!'|'a'): 0.0006377929397804805
p('?'|'a'): 0.0002669830910708988 6
p('/'|'a'): 0.0
p('.'|'a'): 0.0025116187085929003
p('''|'a'): 0.0
p('"'|'a'): 0.0
p('('|'a'): 0.0
p(')'|'a'): 4.449718184514981e-05
p('['|'a'): 0.0
p(']'|'a'): 0.0
p('*'|'a'): 0.0
p('0'|'a'): 0.0
p('1'|'a'): 0.0
p('2'|'a'): 0.0
p('3'|'a'): 0.0
p('4'|'a'): 0.0
p('5'|'a'): 0.0
p('6'|'a'): 0.0
p('7'|'a'): 0.0
p('8'|'a'): 0.0
p('9'|'a'): 0.0
p('a'|'a'): 3.4608919212894296e-05
p('b'|'a'): 0.01710175022248591
p('c'|'a'): 0.034673192920003953
p('d'|'a'): 0.05567586275091466
p('e'|'a'): 0.00033620092949668745
p('f'|'a'): 0.008227034510036587
p('g'|'a'): 0.016691387323247306
p('h'|'a'): 0.0015524572332641154
p('i'|'a'): 0.04310293681400178
p('j'|'a'): 0.0008306140611094631
p('k'|'a'): 0.012523484623751607
p('l'|'a'): 0.07027588252743992
p('m'|'a'): 0.022733115791555424
p('n'|'a'): 0.2246366063482646
p('o'|'a'): 0.00020765351527736578
p('p'|'a'): 0.02289627212498764
p('q'|'a'): 9.888262632255513e-06
p('r'|'a'): 0.08741718580045486
p('s'|'a'): 0.09649461089686542
p('t'|'a'): 0.13835657075051913
p('u'|'a'): 0.012414713734796795
p('v'|'a'): 0.021106496588549392
p('w'|'a'): 0.010669435380203698
p('x'|'a'): 0.00025215069712251556
p('y'|'a'): 0.0257391476317611
p('z'|'a'): 0.0017106694353802036
```

```
Symbol: b
Marginal Probability p('b'): 0.010967599234702054
Transition Probabilities:
p('='|'b'): 0.0
p(' '|'b'): 0.003414253059807297
p('-'|'b'): 0.00014467173982234312
p(','|'b'): 0.0010127021787564017
p(';'|'b'): 0.0
p(':'|'b'): 0.0
p('!'|'b'): 2.893434796446862e-05
p('?'|'b'): 5.786869592893724e-05
p('/'|'b'): 0.0
p('.'|'b'): 0.0005497526113249038
p('''|'b'): 0.0
p('"'|'b'): 0.0
p('('|'b'): 0.0
p(')'|'b'): 5.786869592893724e-05
p('['|'b'): 0.0
p(']'|'b'): 0.0
p('*'|'b'): 0.0
p('0'|'b'): 0.0
p('1'|'b'): 0.0
p('2'|'b'): 0.0
p('3'|'b'): 0.0
p('4'|'b'): 0.0
p('5'|'b'): 0.0
p('6'|'b'): 0.0
p('7'|'b'): 0.0
p('8'|'b'): 0.0
p('9'|'b'): 0.0
p('a'|'b'): 0.09224270131072597
p('b'|'b'): 0.006625965683863314
p('c'|'b'): 8.680304389340587e-05
p('d'|'b'): 0.00046294956743149794
p('e'|'b'): 0.3286363241804346
p('f'|'b'): 0.0
p('g'|'b'): 0.0
p('h'|'b'): 2.893434796446862e-05
p('i'|'b'): 0.029079019704290963
p('j'|'b'): 0.005063510893782009
p('k'|'b'): 0.0
p('l'|'b'): 0.10494488006712768
p('m'|'b'): 0.001996470009548335
p('n'|'b'): 0.00023147478371574897
p('o'|'b'): 0.11831254882671219
```

```
p('p'|'b'): 0.0
p('q'|'b'): 0.0
p('r'|'b'): 0.05723214027371893
p('s'|'b'): 0.013888487022944937
p('t'|'b'): 0.007436127426868435
p('u'|'b'): 0.15262868551257197
p('v'|'b'): 0.0013309800063655566
p('w'|'b'): 0.0005786869592893725
p('x'|'b'): 0.0
p('y'|'b'): 0.07392725904921732
p('z'|'b'): 0.0


Symbol: c
Marginal Probability p('c'): 0.019547529806984088
Transition Probabilities:
p('='|'c'): 0.0
p(' '|'c'): 0.008474301113672521
p('-'|'c'): 3.2468586642423456e-05
p(','|'c'): 0.0006169031462060457
p(';'|'c'): 4.8702879963635184e-05
p(':'|'c'): 1.6234293321211728e-05
p('!'|'c'): 4.8702879963635184e-05
p('?'|'c'): 1.6234293321211728e-05
p('/'|'c'): 0.0
p('.'|'c'): 0.0004870287996363518 4
p('''|'c'): 0.0
p('"'|'c'): 0.0
p('('|'c'): 0.0
p(')'|'c'): 4.8702879963635184e-05
p('['|'c'): 0.0
p(']'|'c'): 0.0
p('*'|'c'): 0.0
p('0'|'c'): 0.0
p('1'|'c'): 0.0
p('2'|'c'): 0.0
p('3'|'c'): 0.0
p('4'|'c'): 0.0
p('5'|'c'): 0.0
p('6'|'c'): 0.0
p('7'|'c'): 0.0
p('8'|'c'): 0.0
p('9'|'c'): 0.0
p('a'|'c'): 0.10912691970518523
p('b'|'c'): 0.0
p('c'|'c'): 0.017029773693951103
p('d'|'c'): 0.00014610863989090555
```

```
p('e'|'c'): 0.21315627130751
p('f'|'c'): 0.0
p('g'|'c'): 0.0
p('h'|'c'): 0.18843144257930453
p('i'|'c'): 0.04173836812883535
p('j'|'c'): 0.0
p('k'|'c'): 0.042631254261501994
p('l'|'c'): 0.03254975810902951
p('m'|'c'): 3.2468586642423456e-05
p('n'|'c'): 0.0
p('o'|'c'): 0.20841585765771617
p('p'|'c'): 0.0
p('q'|'c'): 0.0018994123185817721
p('r'|'c'): 0.03362122146822949
p('s'|'c'): 0.001558492158836326
p('t'|'c'): 0.06720997434981656
p('u'|'c'): 0.02743595571284782
p('v'|'c'): 0.0
p('w'|'c'): 0.0001136400532484821
p('x'|'c'): 0.0
p('y'|'c'): 0.005032630929575636
p('z'|'c'): 8.117146660605864e-05


Symbol: d
Marginal Probability p('d'): 0.03730081737349466
Transition Probabilities:
p('='|'d'): 0.0
p(' '|'d'): 0.5782188494325433
p('-'|'d'): 0.0018971941944156132
p(','|'d'): 0.035646832621531024
p(';'|'d'): 0.0010038964795562437
p(':'|'d'): 0.0015824130948937402
p('!'|'d'): 0.002688400741862483
p('?'|'d'): 0.0015824130948937402
p('/'|'d'): 0.0
p('.'|'d'): 0.0240084395365061
p('''|'d'): 0.0
p('"'|'d'): 8.507597284374947e-06
p('('|'d'): 0.0
p(')'|'d'): 0.0003998570723656225
p('['|'d'): 0.0
p(']'|'d'): 0.0
p('*'|'d'): 0.0
p('0'|'d'): 0.0
p('1'|'d'): 0.0
p('2'|'d'): 0.0
```

```
p('3'|'d'): 0.0
p('4'|'d'): 0.0
p('5'|'d'): 0.0
p('6'|'d'): 0.0
p('7'|'d'): 0.0
p('8'|'d'): 0.0
p('9'|'d'): 0.0
p('a'|'d'): 0.020750029776590496
p('b'|'d'): 0.00010209116741249936
p('c'|'d'): 8.507597284374947e-05
p('d'|'d'): 0.011085399261540556
p('e'|'d'): 0.11691990947916489
p('f'|'d'): 0.0007231457691718705
p('g'|'d'): 0.003692297221418727
p('h'|'d'): 0.0004253798642187473
p('i'|'d'): 0.06630821323441834
p('j'|'d'): 0.0020077929591124873
p('k'|'d'): 0.0005870242126218713
p('l'|'d'): 0.01054091303534056
p('m'|'d'): 0.0026713855472937334
p('n'|'d'): 0.0024672032124687347
p('o'|'d'): 0.04194245461196849
p('p'|'d'): 3.403038913749979e-05
p('q'|'d'): 0.0003828418777968726
p('r'|'d'): 0.030397645097071685
p('s'|'d'): 0.018946419152303006
p('t'|'d'): 0.0003998570723656225
p('u'|'d'): 0.009366864610096816
p('v'|'d'): 0.002722431130999983
p('w'|'d'): 0.00037433428051249765
p('x'|'d'): 0.0
p('y'|'d'): 0.010013442003709313
p('z'|'d'): 1.7015194568749894e-05


Symbol: e
Marginal Probability p('e'): 0.09907745991912265
Transition Probabilities:
p('='|'e'): 0.0
p(' '|'e'): 0.31652210677360254
p('-'|'e'): 0.0009000294671569318
p(','|'e'): 0.019499570804453383
p(';'|'e'): 0.0005092693426261642
p(':'|'e'): 0.0005060663907857482
p('!'|'e'): 0.0021716013478021343
p('?'|'e'): 0.0018449002600796894
p('/'|'e'): 0.0
```

```
p('.'|'e'): 0.01401932020550139
p(''|'e'): 3.2029518404161277e-06
p('"'|'e'): 0.0
p('('|'e'): 0.0
p(')'|'e'): 0.0002338154843503773
p('['|'e'): 0.0
p(']'|'e'): 0.0
p('*'|'e'): 0.0
p('0'|'e'): 0.0
p('1'|'e'): 0.0
p('2'|'e'): 0.0
p('3'|'e'): 0.0
p('4'|'e'): 0.0
p('5'|'e'): 0.0
p('6'|'e'): 0.0
p('7'|'e'): 0.0
p('8'|'e'): 0.0
p('9'|'e'): 0.0
p('a'|'e'): 0.04271456574378948
p('b'|'e'): 0.0008135497674656964
p('c'|'e'): 0.017215866142236684
p('d'|'e'): 0.09163324920246499
p('e'|'e'): 0.02545065532394655
p('f'|'e'): 0.010143748478597876
p('g'|'e'): 0.006860722842171345
p('h'|'e'): 0.002286907614057115
p('i'|'e'): 0.011101431078882298
p('j'|'e'): 0.00027225090643537086
p('k'|'e'): 0.0008583910932315221
p('l'|'e'): 0.03253878774678744
p('m'|'e'): 0.022750566922475755
p('n'|'e'): 0.08393335297810463
p('o'|'e'): 0.005163158366750797
p('p'|'e'): 0.010976515957106069
p('q'|'e'): 0.0008167527193061125
p('r'|'e'): 0.14110924628137292
p('s'|'e'): 0.06281308854240068
p('t'|'e'): 0.02367942295619643
p('u'|'e'): 0.0009961180223694156
p('v'|'e'): 0.016985253609726723
p('w'|'e'): 0.010191792756204118
p('x'|'e'): 0.010038051067864144
p('y'|'e'): 0.01190216903898633
p('z'|'e'): 0.0005445018128707417


Symbol: f
```

```
Marginal Probability p('f'): 0.01738580746136937
Transition Probabilities:
p('='|'f'): 0.0
p(' '|'f'): 0.3353776512247654
p('-'|'f'): 0.0030664768371481766
p(','|'f'): 0.009071660643230022
p(';'|'f'): 0.00016427554484722376
p(':'|'f'): 0.0003468039280108057
p('!'|'f'): 0.0009491475924506261
p('?'|'f'): 0.0005475851494907458
p('/'|'f'): 0.0
p('.'|'f'): 0.007611433577921367
p('''|'f'): 0.0
p('"'|'f'): 0.0
p('('|'f'): 0.0
p(')'|'f'): 0.00021903405979629831
p('['|'f'): 0.0
p(']'|'f'): 0.0
p('*'|'f'): 0.0
p('0'|'f'): 0.0
p('1'|'f'): 0.0
p('2'|'f'): 0.0
p('3'|'f'): 0.0
p('4'|'f'): 0.0
p('5'|'f'): 0.0
p('6'|'f'): 0.0
p('7'|'f'): 0.0
p('8'|'f'): 0.0
p('9'|'f'): 0.0
p('a'|'f'): 0.07032818603292812
p('b'|'f'): 0.0004928266345416712
p('c'|'f'): 0.0
p('d'|'f'): 0.0
p('e'|'f'): 0.08359799948892052
p('f'|'f'): 0.05172854378855912
p('g'|'f'): 0.0
p('h'|'f'): 5.475851494907458e-05
p('i'|'f'): 0.08514949074581098
p('j'|'f'): 0.0
p('k'|'f'): 0.00012776986821450736
p('l'|'f'): 0.022067681524477058
p('m'|'f'): 3.6505676632716386e-05
p('n'|'f'): 9.126419158179097e-05
p('o'|'f'): 0.14768371481765413
p('p'|'f'): 0.0
p('q'|'f'): 0.0
p('r'|'f'): 0.10331106487058737
```

```
p('s'|'f'): 0.0027196729091373707
p('t'|'f'): 0.03924360238017011
p('u'|'f'): 0.0339685321067426
p('v'|'f'): 0.0
p('w'|'f'): 0.0004928266345416712
p('x'|'f'): 0.0
p('y'|'f'): 0.0015514912568904465
p('z'|'f'): 0.0


Symbol: g
Marginal Probability p('g'): 0.016216725676101513
Transition Probabilities:
p('='|'g'): 0.0
p(' '|'g'): 0.3759344056984071
p('-'|'g'): 0.0020155766897577395
p(','|'g'): 0.028668153888301826
p(';'|'g'): 0.0008414543462095417
p(':'|'g'): 0.0011741223435481978
p('!'|'g'): 0.0026809126844350513
p('?'|'g'): 0.0026809126844350513
p('/'|'g'): 7.827482290321318e-05
p('.'|'g'): 0.020625415834996674
p('''|'g'): 0.0
p('"'|'g'): 9.784352862901647e-05
p('('|'g'): 0.0
p(')'|'g'): 0.0004109428202418692
p('['|'g'): 0.0
p(']'|'g'): 0.0
p('*'|'g'): 0.0
p('0'|'g'): 0.0
p('1'|'g'): 0.0
p('2'|'g'): 0.0
p('3'|'g'): 0.0
p('4'|'g'): 0.0
p('5'|'g'): 0.0
p('6'|'g'): 0.0
p('7'|'g'): 0.0
p('8'|'g'): 0.0
p('9'|'g'): 0.0
p('a'|'g'): 0.06573128253297326
p('b'|'g'): 0.0
p('c'|'g'): 0.0
p('d'|'g'): 0.0008218856404837384
p('e'|'g'): 0.11408555438143321
p('f'|'g'): 0.0
p('g'|'g'): 0.008708074047982467
```

```
p('h'|'g'): 0.11664905483151344
p('i'|'g'): 0.04882392078587922
p('j'|'g'): 0.0
p('k'|'g'): 0.0
p('l'|'g'): 0.028805134828382453
p('m'|'g'): 0.0003522367030644593
p('n'|'g'): 0.018120621502093853
p('o'|'g'): 0.06034988845837736
p('p'|'g'): 0.0
p('q'|'g'): 0.0
p('r'|'g'): 0.05363782239442683
p('s'|'g'): 0.016868224335642442
p('t'|'g'): 0.00332667997338656
p('u'|'g'): 0.02626120308402802
p('v'|'g'): 0.0
p('w'|'g'): 0.0003522367030644593
p('x'|'g'): 0.0
p('y'|'g'): 0.0018590270439513132
p('z'|'g'): 3.913741145160659e-05


Symbol: h
Marginal Probability p('h'): 0.05282066367922478
Transition Probabilities:
p('='|'h'): 0.0
p(' '|'h'): 0.07947226761511103
p('-'|'h'): 0.0002222916466403922
p(','|'h'): 0.007636018456214554
p(';'|'h'): 0.00018023647024896665
p(':'|'h'): 8.41103527828511e-05
p('!'|'h'): 0.0011775449389599154
p('?'|'h'): 0.0006608670575795443
p('/'|'h'): 0.0
p('.'|'h'): 0.0032923195232144576
p('''|'h'): 0.0
p('"'|'h'): 0.0
p('('|'h'): 0.0
p(')'|'h'): 6.608670575795444e-05
p('['|'h'): 0.0
p(']'|'h'): 0.0
p('*'|'h'): 0.0
p('0'|'h'): 0.0
p('1'|'h'): 0.0
p('2'|'h'): 0.0
p('3'|'h'): 0.0
p('4'|'h'): 0.0
p('5'|'h'): 0.0
```

```
p('6'|'h'): 0.0
p('7'|'h'): 0.0
p('8'|'h'): 0.0
p('9'|'h'): 0.0
p('a'|'h'): 0.1649223781601461
p('b'|'h'): 0.00037248870518119775
p('c'|'h'): 0.00037248870518119775
p('d'|'h'): 0.0003604729404979333
p('e'|'h'): 0.4499603479765452
p('f'|'h'): 0.00040853599923099105
p('g'|'h'): 6.007882341632221e-06
p('h'|'h'): 3.6047294049793326e-05
p('i'|'h'): 0.15511751417860234
p('j'|'h'): 0.0
p('k'|'h'): 0.000570748822455061
p('l'|'h'): 0.0009011823512448332
p('m'|'h'): 0.0022649716427953475
p('n'|'h'): 0.0006969143516293377
p('o'|'h'): 0.08218783043352879
p('p'|'h'): 2.4031529366528884e-05
p('q'|'h'): 0.0
p('r'|'h'): 0.007696097279630876
p('s'|'h'): 0.0009432375276362588
p('t'|'h'): 0.025191050658463904
p('u'|'h'): 0.009570556570220129
p('v'|'h'): 2.4031529366528884e-05
p('w'|'h'): 0.00021628376429875997
p('x'|'h'): 0.0
p('y'|'h'): 0.0053650389310775735
p('z'|'h'): 0.0


Symbol: i
Marginal Probability p('i'): 0.05444830224508765
Transition Probabilities:
p('='|'i'): 0.0
p(' '|'i'): 0.02912977846681082
p('-'|'i'): 0.00012822231417963946
p(','|'i'): 0.0012938797158127254
p(';'|'i'): 1.748486102449629e-05
p(':'|'i'): 5.82828700816543e-06
p('!'|'i'): 0.00012239402717147404
p('?'|'i'): 0.00024478805434294807
p('/'|'i'): 0.0
p('.'|'i'): 0.0007518490240533404
p('''|'i'): 0.0
p('"'|'i'): 1.165657401633086e-05
```

```
p('('|'i'): 0.0
p(')'|'i'): 0.0
p('['|'i'): 0.0
p(']'|'i'): 0.0
p('*'|'i'): 0.0
p('0'|'i'): 0.0
p('1'|'i'): 0.0
p('2'|'i'): 0.0
p('3'|'i'): 0.0
p('4'|'i'): 0.0
p('5'|'i'): 0.0
p('6'|'i'): 0.0
p('7'|'i'): 0.0
p('8'|'i'): 0.0
p('9'|'i'): 0.0
p('a'|'i'): 0.016954486906753236
p('b'|'i'): 0.007413581074386427
p('c'|'i'): 0.04960455072649598
p('d'|'i'): 0.050385541185590146
p('e'|'i'): 0.04920239892293256
p('f'|'i'): 0.021832763132587703
p('g'|'i'): 0.02506163413511135
p('h'|'i'): 3.496972204899258e-05
p('i'|'i'): 0.0026810120237560978
p('j'|'i'): 0.0
p('k'|'i'): 0.006982287835782185
p('l'|'i'): 0.04617168967868654
p('m'|'i'): 0.05719880869813553
p('n'|'i'): 0.2806611608782063
p('o'|'i'): 0.04237747483637084
p('p'|'i'): 0.004843306503785472
p('q'|'i'): 0.0001573637492204666
p('r'|'i'): 0.03445100450526586
p('s'|'i'): 0.12304096702938039
p('t'|'i'): 0.12318667420458453
p('u'|'i'): 0.0007576773110615059
p('v'|'i'): 0.02002016587304825
p('w'|'i'): 2.331314803266172e-05
p('x'|'i'): 0.0018883649906455994
p('y'|'i'): 5.82828700816543e-06
p('z'|'i'): 0.0033570933167032875


Symbol: j
Marginal Probability p('j'): 0.0008158819950932838
Transition Probabilities:
p('='|'j'): 0.0
```

```
p(' '|'j'): 0.0
p('-'|'j'): 0.0
p(','|'j'): 0.0
p(';'|'j'): 0.0
p(':'|'j'): 0.0
p('!'|'j'): 0.0
p('?'|'j'): 0.0
p('/'|'j'): 0.0
p('.'|'j'): 0.0
p('''|'j'): 0.0
p('"'|'j'): 0.0
p('('|'j'): 0.0
p(')'|'j'): 0.0
p('['|'j'): 0.0
p(']'|'j'): 0.0
p('*'|'j'): 0.0
p('0'|'j'): 0.0
p('1'|'j'): 0.0
p('2'|'j'): 0.0
p('3'|'j'): 0.0
p('4'|'j'): 0.0
p('5'|'j'): 0.0
p('6'|'j'): 0.0
p('7'|'j'): 0.0
p('8'|'j'): 0.0
p('9'|'j'): 0.0
p('a'|'j'): 0.03695060287825749
p('b'|'j'): 0.0
p('c'|'j'): 0.0
p('d'|'j'): 0.0
p('e'|'j'): 0.21975884869700504
p('f'|'j'): 0.0
p('g'|'j'): 0.0
p('h'|'j'): 0.0
p('i'|'j'): 0.004278490859587709
p('j'|'j'): 0.0
p('k'|'j'): 0.0
p('l'|'j'): 0.0
p('m'|'j'): 0.0
p('n'|'j'): 0.0
p('o'|'j'): 0.2784908595877091
p('p'|'j'): 0.0
p('q'|'j'): 0.0
p('r'|'j'): 0.0
p('s'|'j'): 0.0
p('t'|'j'): 0.0
p('u'|'j'): 0.4605211979774407
```

```
p('v'|'j'): 0.0
p('w'|'j'): 0.0
p('x'|'j'): 0.0
p('y'|'j'): 0.0
p('z'|'j'): 0.0


Symbol: k
Marginal Probability p('k'): 0.006325544849550535
Transition Probabilities:
p('='|'k'): 0.0
p(' '|'k'): 0.19906687402799378
p('-'|'k'): 0.0016053780163547886
p(','|'k'): 0.027843274971153364
p(';'|'k'): 0.0008026890081773943
p(':'|'k'): 0.0005016806301108714
p('!'|'k'): 0.003060251843676316
p('?'|'k'): 0.0026087392765765313
p('/'|'k'): 5.016806301108714e-05
p('.'|'k'): 0.019565544574323985
p('''|'k'): 0.0
p('"'|'k'): 0.0
p('('|'k'): 0.0
p(')'|'k'): 0.0003010083780665229
p('['|'k'): 0.0
p(']'|'k'): 0.0
p('*'|'k'): 0.0
p('0'|'k'): 0.0
p('1'|'k'): 0.0
p('2'|'k'): 0.0
p('3'|'k'): 0.0
p('4'|'k'): 0.0
p('5'|'k'): 0.0
p('6'|'k'): 0.0
p('7'|'k'): 0.0
p('8'|'k'): 0.0
p('9'|'k'): 0.0
p('a'|'k'): 0.025234535694576832
p('b'|'k'): 0.00035117644107761
p('c'|'k'): 0.0023578989615210956
p('d'|'k'): 0.00015050418903326144
p('e'|'k'): 0.2836502282646867
p('f'|'k'): 0.0006521848191441328
p('g'|'k'): 0.0006020167561330457
p('h'|'k'): 0.040987307480058195
p('i'|'k'): 0.16981889329252997
p('j'|'k'): 0.0
```

```
p('k'|'k'): 0.0
p('l'|'k'): 0.017006973360758543
p('m'|'k'): 0.001454873827321527
p('n'|'k'): 0.10389805849596147
p('o'|'k'): 0.012792856067827221
p('p'|'k'): 5.016806301108714e-05
p('q'|'k'): 0.0
p('r'|'k'): 0.004765965986053279
p('s'|'k'): 0.03742537500627101
p('t'|'k'): 0.0002508403150554357
p('u'|'k'): 0.03341192996538404
p('v'|'k'): 0.0006020167561330457
p('w'|'k'): 0.004113781166909146
p('x'|'k'): 0.0
p('y'|'k'): 0.005016806301108714
p('z'|'k'): 0.0


Symbol: l
Marginal Probability p('l'): 0.03037010450969173
Transition Probabilities:
p('='|'l'): 0.0
p(' '|'l'): 0.11103216233725523
p('-'|'l'): 0.0011702994712754174
p(','|'l'): 0.014566048776410106
p(';'|'l'): 0.000355269482351466
p(':'|'l'): 0.00021943115086414078
p('!'|'l'): 0.0014942216463605777
p('?'|'l'): 0.0007627844768134418
p('/'|'l'): 0.0
p('.'|'l'): 0.006300808760527471
p('''|'l'): 0.0
p('"'|'l'): 0.0
p('('|'l'): 0.0
p(')'|'l'): 0.00013583833148732524
p('['|'l'): 0.0
p(']'|'l'): 0.0
p('*'|'l'): 0.0
p('0'|'l'): 0.0
p('1'|'l'): 0.0
p('2'|'l'): 0.0
p('3'|'l'): 0.0
p('4'|'l'): 0.0
p('5'|'l'): 0.0
p('6'|'l'): 0.0
p('7'|'l'): 0.0
p('8'|'l'): 0.0
```

```
p('9'|'l'): 0.0
p('a'|'l'): 0.07928778917890954
p('b'|'l'): 0.0007418862719692378
p('c'|'l'): 0.0007314371695471359
p('d'|'l'): 0.07223464504399073
p('e'|'l'): 0.17507471108231804
p('f'|'l'): 0.024074731980522873
p('g'|'l'): 0.000898622808300767
p('h'|'l'): 4.179640968840777e-05
p('i'|'l'): 0.09829470648471296
p('j'|'l'): 0.0
p('k'|'l'): 0.01082527010929761
p('l'|'l'): 0.13803264299596665
p('m'|'l'): 0.004096048149463961
p('n'|'l'): 0.0012747904954964369
p('o'|'l'): 0.0866439572840693
p('p'|'l'): 0.003970658920398737
p('q'|'l'): 1.0449102422101942e-05
p('r'|'l'): 0.0048379344214331986
p('s'|'l'): 0.016237905163946416
p('t'|'l'): 0.017157426177091387
p('u'|'l'): 0.015412426072600363
p('v'|'l'): 0.004419970324549122
p('w'|'l'): 0.004879730831121607
p('x'|'l'): 0.0
p('y'|'l'): 0.10452237152828572
p('z'|'l'): 0.00026122756055254855


Symbol: m
Marginal Probability p('m'): 0.019334911784147646
Transition Probabilities:
p('='|'m'): 0.0
p(' '|'m'): 0.14845391281512604
p('-'|'m'): 0.00011488970588235294
p(','|'m'): 0.023585215336134453
p(';'|'m'): 0.0010340073529411765
p(':'|'m'): 0.0007385766806722689
p('!'|'m'): 0.001953125
p('?'|'m'): 0.0016905199579831933
p('/'|'m'): 0.0
p('.'|'m'): 0.02696625525210084
p('''|'m'): 1.641281512605042e-05
p('"'|'m'): 0.0
p('('|'m'): 0.0
p(')'|'m'): 0.0004267331932773109
p('['|'m'): 0.0
```

```
p(']'|'m'): 0.0
p('*'|'m'): 0.0
p('0'|'m'): 0.0
p('1'|'m'): 0.0
p('2'|'m'): 0.0
p('3'|'m'): 0.0
p('4'|'m'): 0.0
p('5'|'m'): 0.0
p('6'|'m'): 0.0
p('7'|'m'): 0.0
p('8'|'m'): 0.0
p('9'|'m'): 0.0
p('a'|'m'): 0.15183495273109243
p('b'|'m'): 0.017807904411764705
p('c'|'m'): 0.0005908613445378151
p('d'|'m'): 0.0
p('e'|'m'): 0.2477842699579832
p('f'|'m'): 0.002182904411764706
p('g'|'m'): 1.641281512605042e-05
p('h'|'m'): 0.0
p('i'|'m'): 0.07910976890756302
p('j'|'m'): 0.0
p('k'|'m'): 9.847689075630252e-05
p('l'|'m'): 0.002658876050420168
p('m'|'m'): 0.02269892331932773
p('n'|'m'): 0.0037257090336134456
p('o'|'m'): 0.10903033088235294
p('p'|'m'): 0.05627954306722689
p('q'|'m'): 0.0
p('r'|'m'): 0.001329438025210084
p('s'|'m'): 0.031496192226890755
p('t'|'m'): 0.0009683560924369748
p('u'|'m'): 0.026999080882352942
p('v'|'m'): 0.0
p('w'|'m'): 9.847689075630252e-05
p('x'|'m'): 0.0
p('y'|'m'): 0.040309873949579834
p('z'|'m'): 0.0


Symbol: n
Marginal Probability p('n'): 0.0576271003566588
Transition Probabilities:
p('='|'n'): 0.0
p(' '|'n'): 0.1924292652840953
p('-'|'n'): 0.0008535524301463705
p(','|'n'): 0.016602971463814883
```

```
p(';'|'n'): 0.000490104298600174
p(':'|'n'): 0.00037996850116193265
p('!'|'n'): 0.0014317653666971375
p('?'|'n'): 0.0010848376047666772
p('/'|'n'): 0.0
p('.'|'n'): 0.01105212727292752
p(''|'n'): 1.6520369615736202e-05
p('"'|'n'): 5.506789871912068e-06
p('('|'n'): 0.0
p(')'|'n'): 0.00021476480500457065
p('['|'n'): 0.0
p(']'|'n'): 0.0
p('*'|'n'): 0.0
p('0'|'n'): 0.0
p('1'|'n'): 0.0
p('2'|'n'): 0.0
p('3'|'n'): 0.0
p('4'|'n'): 0.0
p('5'|'n'): 0.0
p('6'|'n'): 0.0
p('7'|'n'): 0.0
p('8'|'n'): 0.0
p('9'|'n'): 0.0
p('a'|'n'): 0.030298357875260196
p('b'|'n'): 0.0008535524301463705
p('c'|'n'): 0.05012280141414364
p('d'|'n'): 0.19035871229225634
p('e'|'n'): 0.08161062590173684
p('f'|'n'): 0.005022192363183806
p('g'|'n'): 0.13764771963831404
p('h'|'n'): 0.0007874709516834257
p('i'|'n'): 0.03128958005220437
p('j'|'n'): 0.0006608147846294481
p('k'|'n'): 0.008210623699020893
p('l'|'n'): 0.011580779100631078
p('m'|'n'): 0.0007599370023238653
p('n'|'n'): 0.01073273346035662
p('o'|'n'): 0.06920382832031895
p('p'|'n'): 0.0005286518277035585
p('q'|'n'): 0.0013381499388746324
p('r'|'n'): 0.000407502450521493
p('s'|'n'): 0.033998920669185105
p('t'|'n'): 0.08808110400123352
p('u'|'n'): 0.005247970747932201
p('v'|'n'): 0.004146612773549787
p('w'|'n'): 0.0005837197264226792
p('x'|'n'): 0.00045155676949678957
```

```
p('y'|'n'): 0.011393548244986068
p('z'|'n'): 0.00012114937718206549


Symbol: o
Marginal Probability p('o'): 0.060287364364775095
Transition Probabilities:
p('='|'o'): 0.0
p(' '|'o'): 0.13310558646573006
p('-'|'o'): 0.00041583981218779113
p(','|'o'): 0.004453170646973054
p(';'|'o'): 7.369313127378577e-05
p(':'|'o'): 7.369313127378577e-05
p('!'|'o'): 0.0007474589029198271
p('?'|'o'): 0.0006421830011001332
p('/'|'o'): 0.0
p('.'|'o'): 0.001868647257299568
p('''|'o'): 0.0
p('"'|'o'): 0.0
p('('|'o'): 0.0
p(')'|'o'): 4.7374155818862286e-05
p('['|'o'): 0.0
p(']'|'o'): 0.0
p('*'|'o'): 0.0
p('0'|'o'): 0.0
p('1'|'o'): 0.0
p('2'|'o'): 0.0
p('3'|'o'): 0.0
p('4'|'o'): 0.0
p('5'|'o'): 0.0
p('6'|'o'): 0.0
p('7'|'o'): 0.0
p('8'|'o'): 0.0
p('9'|'o'): 0.0
p('a'|'o'): 0.007258773430467899
p('b'|'o'): 0.004684777630976382
p('c'|'o'): 0.00724824584028593
p('d'|'o'): 0.016059838822594315
p('e'|'o'): 0.002631897545492349
p('f'|'o'): 0.08906867673455207
p('g'|'o'): 0.0040531222200582175
p('h'|'o'): 0.0018949662327544913
p('i'|'o'): 0.011754054438168831
p('j'|'o'): 0.0006421830011001332
p('k'|'o'): 0.017754780841891388
p('l'|'o'): 0.03875205945982935
p('m'|'o'): 0.058207046116108795
```

```
p('n'|'o'): 0.1409381135611153
p('o'|'o'): 0.03788879706490786
p('p'|'o'): 0.01610194918332219
p('q'|'o'): 0.00013159487727461747
p('r'|'o'): 0.10392837027640188
p('s'|'o'): 0.035509561683782775
p('t'|'o'): 0.05277480958221258
p('u'|'o'): 0.12394658300741669
p('v'|'o'): 0.03018260105170626
p('w'|'o'): 0.05216420935165836
p('x'|'o'): 0.0007421951078288424
p('y'|'o'): 0.003658337588234365
p('z'|'o'): 0.0005948088452812709


Symbol: p
Marginal Probability p('p'): 0.014238426042724798
Transition Probabilities:
p('='|'p'): 0.0
p(' '|'p'): 0.05402514041187483
p('-'|'p'): 0.00015601319425871446
p(','|'p'): 0.00849157528751003
p(';'|'p'): 0.0001783007934385308
p(':'|'p'): 0.00011143799589908175
p('!'|'p'): 0.0008469287688330213
p('?'|'p'): 0.0003566015868770616
p('/'|'p'): 0.0
p('.'|'p'): 0.006039939377730231
p('''|'p'): 0.0
p('"'|'p'): 0.0
p('('|'p'): 0.0
p(')'|'p'): 8.91503967192654e-05
p('['|'p'): 0.0
p(']'|'p'): 0.0
p('*'|'p'): 0.0
p('0'|'p'): 0.0
p('1'|'p'): 0.0
p('2'|'p'): 0.0
p('3'|'p'): 0.0
p('4'|'p'): 0.0
p('5'|'p'): 0.0
p('6'|'p'): 0.0
p('7'|'p'): 0.0
p('8'|'p'): 0.0
p('9'|'p'): 0.0
p('a'|'p'): 0.09826602478381029
p('b'|'p'): 0.00024516359097797983
```

```
p('c'|'p'): 0.0028305250958366767
p('d'|'p'): 4.45751983596327e-05
p('e'|'p'): 0.18249086208433626
p('f'|'p'): 0.002228759917981635
p('g'|'p'): 4.45751983596327e-05
p('h'|'p'): 0.009249353659623785
p('i'|'p'): 0.09365249175358831
p('j'|'p'): 0.0
p('k'|'p'): 0.0004011767852366943
p('l'|'p'): 0.09405366853882499
p('m'|'p'): 0.0006017651778550414
p('n'|'p'): 0.0001783007934385308
p('o'|'p'): 0.10568779531068913
p('p'|'p'): 0.05850494784701792
p('q'|'p'): 0.0
p('r'|'p'): 0.17524739235089595
p('s'|'p'): 0.023914593919942943
p('t'|'p'): 0.04831951502184185
p('u'|'p'): 0.025809039850227332
p('v'|'p'): 0.0
p('w'|'p'): 0.00042346438441651064
p('x'|'p'): 0.0
p('y'|'p'): 0.00751092092359811
p('z'|'p'): 0.0


Symbol: q
Marginal Probability p('q'): 0.0007397203152712736
Transition Probabilities:
p('='|'q'): 0.0
p(' '|'q'): 0.000429000429000429
p('-'|'q'): 0.0
p(','|'q'): 0.000429000429000429
p(';'|'q'): 0.0
p(':'|'q'): 0.0
p('!'|'q'): 0.0
p('?'|'q'): 0.0
p('/'|'q'): 0.0
p('.'|'q'): 0.0
p('''|'q'): 0.0
p('"'|'q'): 0.0
p('('|'q'): 0.0
p(')'|'q'): 0.0
p('['|'q'): 0.0
p(']'|'q'): 0.0
p('*'|'q'): 0.0
p('0'|'q'): 0.0
```

```
p('1'|'q'): 0.0
p('2'|'q'): 0.0
p('3'|'q'): 0.0
p('4'|'q'): 0.0
p('5'|'q'): 0.0
p('6'|'q'): 0.0
p('7'|'q'): 0.0
p('8'|'q'): 0.0
p('9'|'q'): 0.0
p('a'|'q'): 0.0
p('b'|'q'): 0.0
p('c'|'q'): 0.0
p('d'|'q'): 0.0
p('e'|'q'): 0.0
p('f'|'q'): 0.0
p('g'|'q'): 0.0
p('h'|'q'): 0.0
p('i'|'q'): 0.0
p('j'|'q'): 0.0
p('k'|'q'): 0.0
p('l'|'q'): 0.0
p('m'|'q'): 0.0
p('n'|'q'): 0.0
p('o'|'q'): 0.0
p('p'|'q'): 0.0
p('q'|'q'): 0.0
p('r'|'q'): 0.0
p('s'|'q'): 0.0
p('t'|'q'): 0.0
p('u'|'q'): 0.9991419991419992
p('v'|'q'): 0.0
p('w'|'q'): 0.0
p('x'|'q'): 0.0
p('y'|'q'): 0.0
p('z'|'q'): 0.0


Symbol: r
Marginal Probability p('r'): 0.04653573839224598
Transition Probabilities:
p('='|'r'): 0.0
p(' '|'r'): 0.17689899961130093
p('-'|'r'): 0.0012138322320192577
p(','|'r'): 0.019230375810642172
p(';'|'r'): 0.0006546510914261165
p(':'|'r'): 0.0004159762143436782
p('!'|'r'): 0.0017730133726123988
```

```
p('?'|'r'): 0.0017730133726123988
p('/'|'r'): 0.0
p('.'|'r'): 0.01427275764952981
p(''|'r'): 0.0
p('"'|'r'): 0.0
p('('|'r'): 0.0
p(')'|'r'): 0.0002386748770824383
p('['|'r'): 0.0
p(']'|'r'): 0.0
p('*'|'r'): 0.0
p('0'|'r'): 0.0
p('1'|'r'): 0.0
p('2'|'r'): 0.0
p('3'|'r'): 0.0
p('4'|'r'): 0.0
p('5'|'r'): 0.0
p('6'|'r'): 0.0
p('7'|'r'): 0.0
p('8'|'r'): 0.0
p('9'|'r'): 0.0
p('a'|'r'): 0.05483384818913961
p('b'|'r'): 0.0014456878268993406
p('c'|'r'): 0.010010706273057697
p('d'|'r'): 0.02968433542685297
p('e'|'r'): 0.24225500023867488
p('f'|'r'): 0.0030550384266552102
p('g'|'r'): 0.00951971795448811
p('h'|'r'): 0.0014388685446969853
p('i'|'r'): 0.0887802349924647
p('j'|'r'): 1.363856440471076e-05
p('k'|'r'): 0.006307836037178727
p('l'|'r'): 0.008599114857170135
p('m'|'r'): 0.019960039006294197
p('n'|'r'): 0.017457362438029773
p('o'|'r'): 0.10025026765682644
p('p'|'r'): 0.003709689518081327
p('q'|'r'): 8.865066863061995e-05
p('r'|'r'): 0.0335917841288026
p('s'|'r'): 0.055045245937412626
p('t'|'r'): 0.0313755174130371
p('u'|'r'): 0.019728183411414114
p('v'|'r'): 0.004807593952660543
p('w'|'r'): 0.0018889411700524403
p('x'|'r'): 0.0
p('y'|'r'): 0.03912222199491282
p('z'|'r'): 0.0005591811405931411
```

```
Symbol: s
Marginal Probability p('s'): 0.05118985171003598
Transition Probabilities:
p('='|'s'): 0.0
p(' '|'s'): 0.32053388217644396
p('-'|'s'): 0.00030996410615650706
p(','|'s'): 0.03767303746226187
p(';'|'s'): 0.0010848743715477747
p(':'|'s'): 0.00087409877936135
p('!'|'s'): 0.003112039625811331
p('?'|'s'): 0.0016490090447526177
p('/'|'s'): 6.199282123130141e-06
p('.'|'s'): 0.021269736964459517
p('''|'s'): 6.199282123130141e-06
p('"'|'s'): 0.0
p('('|'s'): 0.0
p(')'|'s'): 0.00047114544135789074
p('['|'s'): 0.0
p(']'|'s'): 0.0
p('*'|'s'): 0.0
p('0'|'s'): 0.0
p('1'|'s'): 0.0
p('2'|'s'): 0.0
p('3'|'s'): 0.0
p('4'|'s'): 0.0
p('5'|'s'): 0.0
p('6'|'s'): 0.0
p('7'|'s'): 0.0
p('8'|'s'): 0.0
p('9'|'s'): 0.0
p('a'|'s'): 0.05199957844881563
p('b'|'s'): 0.0026346949023303102
p('c'|'s'): 0.013799602006087696
p('d'|'s'): 0.0003905547737571989
p('e'|'s'): 0.10877260413244147
p('f'|'s'): 0.0018597846369390425
p('g'|'s'): 0.0002727684134177262
p('h'|'s'): 0.06720641749685387
p('i'|'s'): 0.05622128957466725
p('j'|'s'): 1.8597846369390424e-05
p('k'|'s'): 0.013353253693222326
p('l'|'s'): 0.0074453378298793
p('m'|'s'): 0.010451989659597418
p('n'|'s'): 0.0027214848520541323
p('o'|'s'): 0.05298526430639332
p('p'|'s'): 0.0200732755146954
```

```
p('q'|'s'): 0.0010476786788089938
p('r'|'s'): 0.00015498205307825353
p('s'|'s'): 0.06002144951614603
p('t'|'s'): 0.11014884476377636
p('u'|'s'): 0.02446236725787154
p('v'|'s'): 0.0004897432877272812
p('w'|'s'): 0.004147319740374065
p('x'|'s'): 0.0
p('y'|'s'): 0.0023061329498044124
p('z'|'s'): 2.4797128492520565e-05


Symbol: t
Marginal Probability p('t'): 0.07049398148192224
Transition Probabilities:
p('='|'t'): 0.0
p(' '|'t'): 0.2319528225443414
p('-'|'t'): 0.00046817322409291437
p(','|'t'): 0.014535878274961736
p(';'|'t'): 0.0004951832177905825
p(':'|'t'): 0.0002746016025929594
p('!'|'t'): 0.0015935896281624202
p('?'|'t'): 0.001971729539929774
p('/'|'t'): 0.0
p('.'|'t'): 0.011119114072206716
p('''|'t'): 0.0
p('"'|'t'): 0.0
p('('|'t'): 0.0
p(')'|'t'): 0.000198073287116233
p('['|'t'): 0.0
p(']'|'t'): 0.0
p('*'|'t'): 0.0
p('0'|'t'): 0.0
p('1'|'t'): 0.0
p('2'|'t'): 0.0
p('3'|'t'): 0.0
p('4'|'t'): 0.0
p('5'|'t'): 0.0
p('6'|'t'): 0.0
p('7'|'t'): 0.0
p('8'|'t'): 0.0
p('9'|'t'): 0.0
p('a'|'t'): 0.03414063203385252
p('b'|'t'): 6.752498424417034e-05
p('c'|'t'): 0.0028540560007202665
p('d'|'t'): 1.3504996848834069e-05
p('e'|'t'): 0.08616638156117763
```

```
p('f'|'t'): 0.0008373098046277123
p('g'|'t'): 4.051499054650221e-05
p('h'|'t'): 0.33208337084721345
p('i'|'t'): 0.06256865040064824
p('j'|'t'): 0.0
p('k'|'t'): 4.501665616278023e-05
p('l'|'t'): 0.013500495183217791
p('m'|'t'): 0.0012334563788601782
p('n'|'t'): 0.0008012964796974881
p('o'|'t'): 0.10678401008373098
p('p'|'t'): 0.0001575582965697308
p('q'|'t'): 0.0
p('r'|'t'): 0.025465922391284774
p('s'|'t'): 0.01755649590348429
p('t'|'t'): 0.01969928873683263
p('u'|'t'): 0.01334293688664806
p('v'|'t'): 7.652831547672639e-05
p('w'|'t'): 0.00523543711173134
p('x'|'t'): 0.0
p('y'|'t'): 0.014103718375799046
p('z'|'t'): 0.0006167281894300891


Symbol: u
Marginal Probability p('u'): 0.020363729142409966
Transition Probabilities:
p('='|'u'): 0.0
p(' '|'u'): 0.04866760168302945
p('-'|'u'): 4.675081813931744e-05
p(','|'u'): 0.004815334268349696
p(';'|'u'): 0.00014025245441795232
p(':'|'u'): 7.791803023219573e-05
p('!'|'u'): 0.0016362786348761104
p('?'|'u'): 0.001496026180458158
p('/'|'u'): 0.0
p('.'|'u'): 0.0030543867851020724
p('''|'u'): 0.0
p('"'|'u'): 0.0
p('('|'u'): 0.0
p(')'|'u'): 4.675081813931744e-05
p('['|'u'): 0.0
p(']'|'u'): 0.0
p('*'|'u'): 0.0
p('0'|'u'): 0.0
p('1'|'u'): 0.0
p('2'|'u'): 0.0
p('3'|'u'): 0.0
```

```
p('4'|'u'): 0.0
p('5'|'u'): 0.0
p('6'|'u'): 0.0
p('7'|'u'): 0.0
p('8'|'u'): 0.0
p('9'|'u'): 0.0
p('a'|'u'): 0.02060152719339255
p('b'|'u'): 0.013978494623655914
p('c'|'u'): 0.03193080878915381
p('d'|'u'): 0.022004051737572073
p('e'|'u'): 0.029499766245909304
p('f'|'u'): 0.005142589995324918
p('g'|'u'): 0.05064671965092722
p('h'|'u'): 0.0006545114539504442
p('i'|'u'): 0.027271310581268506
p('j'|'u'): 1.5583606046439145e-05
p('k'|'u'): 0.001496026180458158
p('l'|'u'): 0.09770920991117345
p('m'|'u'): 0.021271622253389435
p('n'|'u'): 0.13662147420913198
p('o'|'u'): 0.002306373694872994
p('p'|'u'): 0.04324450677886863
p('q'|'u'): 1.5583606046439145e-05
p('r'|'u'): 0.12457534673523453
p('s'|'u'): 0.13691756272401434
p('t'|'u'): 0.1707028206326944
p('u'|'u'): 0.0
p('v'|'u'): 0.0012466884837151317
p('w'|'u'): 3.116721209287829e-05
p('x'|'u'): 0.0006700950599968833
p('y'|'u'): 0.0003116721209287829
p('z'|'u'): 0.001153186847436497


Symbol: v
Marginal Probability p('v'): 0.008339703940510111
Transition Probabilities:
p('='|'v'): 0.0
p(' '|'v'): 0.06643835616438357
p('-'|'v'): 0.00041856925418569255
p(','|'v'): 0.021575342465753426
p(';'|'v'): 0.00041856925418569255
p(':'|'v'): 0.00022831050228310502
p('!'|'v'): 0.00106544901065449
p('?'|'v'): 0.0009512937595129375
p('/'|'v'): 0.0
p('.'|'v'): 0.010616438356164383
```

```
p(''|'v'): 0.0
p('"'|'v'): 0.0
p('('|'v'): 0.0
p(')'|'v'): 0.00022831050228310502
p('['|'v'): 0.0
p(']'|'v'): 0.0
p('*'|'v'): 0.0
p('0'|'v'): 0.0
p('1'|'v'): 0.0
p('2'|'v'): 0.0
p('3'|'v'): 0.0
p('4'|'v'): 0.0
p('5'|'v'): 0.0
p('6'|'v'): 0.0
p('7'|'v'): 0.0
p('8'|'v'): 0.0
p('9'|'v'): 0.0
p('a'|'v'): 0.06430745814307458
p('b'|'v'): 0.0
p('c'|'v'): 0.0
p('d'|'v'): 0.0
p('e'|'v'): 0.5516742770167428
p('f'|'v'): 0.0
p('g'|'v'): 0.00011415525114155251
p('h'|'v'): 0.0
p('i'|'v'): 0.17937595129375952
p('j'|'v'): 0.0
p('k'|'v'): 0.00030441400304414006
p('l'|'v'): 0.008181126331811263
p('m'|'v'): 0.0
p('n'|'v'): 0.020243531202435314
p('o'|'v'): 0.05570776255707763
p('p'|'v'): 0.0
p('q'|'v'): 0.0
p('r'|'v'): 0.003310502283105023
p('s'|'v'): 0.00993150684931507
p('t'|'v'): 7.610350076103501e-05
p('u'|'v'): 0.00076103500761035
p('v'|'v'): 3.805175038051751e-05
p('w'|'v'): 3.805175038051751e-05
p('x'|'v'): 0.0
p('y'|'v'): 0.003995433789954338
p('z'|'v'): 0.0


Symbol: w
Marginal Probability p('w'): 0.01873037845056044
```

```
Transition Probabilities:
p('='|'w'): 0.0
p(' '|'w'): 0.11048235433644511
p('-'|'w'): 0.0003727360520475069
p(','|'w'): 0.01650204157701235
p(';'|'w'): 0.0004405062433288718
p(':'|'w'): 0.00025413821730511834
p('!'|'w'): 0.0015417718516510513
p('?'|'w'): 0.001440116564729004
p('/'|'w'): 0.0
p('.'|'w'): 0.009132033275163919
p('''|'w'): 0.0
p('"'|'w'): 0.0
p('('|'w'): 0.0
p(')'|'w'): 0.0002880233129458008
p('['|'w'): 0.0
p(']'|'w'): 0.0
p('*'|'w'): 0.0
p('0'|'w'): 0.0
p('1'|'w'): 0.0
p('2'|'w'): 0.0
p('3'|'w'): 0.0
p('4'|'w'): 0.0
p('5'|'w'): 0.0
p('6'|'w'): 0.0
p('7'|'w'): 0.0
p('8'|'w'): 0.0
p('9'|'w'): 0.0
p('a'|'w'): 0.20613997933009165
p('b'|'w'): 6.777019128136489e-05
p('c'|'w'): 0.00027108076512545956
p('d'|'w'): 0.006217915050065229
p('e'|'w'): 0.13142334344238688
p('f'|'w'): 0.0007454721040950138
p('g'|'w'): 0.0
p('h'|'w'): 0.2020907104010301
p('i'|'w'): 0.1697812717076394
p('j'|'w'): 0.0
p('k'|'w'): 0.001067380512681497
p('l'|'w'): 0.0033715670162479034
p('m'|'w'): 3.3885095640682445e-05
p('n'|'w'): 0.03762939870897786
p('o'|'w'): 0.0778848923301086
p('p'|'w'): 1.6942547820341222e-05
p('q'|'w'): 0.0
p('r'|'w'): 0.008945665249140165
p('s'|'w'): 0.012368059908849092
```

```
p('t'|'w'): 0.00011859783474238856
p('u'|'w'): 0.00032190840858648326
p('v'|'w'): 0.0
p('w'|'w'): 0.0003896785998678481
p('x'|'w'): 0.0
p('y'|'w'): 0.0006607593649933077
p('z'|'w'): 0.0


Symbol: x
Marginal Probability p('x'): 0.0013747183207872833
Transition Probabilities:
p('='|'x'): 0.0
p(' '|'x'): 0.07156048014773776
p('-'|'x'): 0.0018467220683287165
p(','|'x'): 0.008541089566020314
p(';'|'x'): 0.0
p(':'|'x'): 0.0006925207756232687
p('!'|'x'): 0.0004616805170821791
p('?'|'x'): 0.0
p('/'|'x'): 0.00023084025854108956
p('.'|'x'): 0.0043859649122807015
p('''|'x'): 0.0
p('"'|'x'): 0.0
p('('|'x'): 0.0
p(')'|'x'): 0.0
p('['|'x'): 0.0
p(']'|'x'): 0.0
p('*'|'x'): 0.0
p('0'|'x'): 0.0
p('1'|'x'): 0.0
p('2'|'x'): 0.0
p('3'|'x'): 0.0
p('4'|'x'): 0.0
p('5'|'x'): 0.0
p('6'|'x'): 0.0
p('7'|'x'): 0.0
p('8'|'x'): 0.0
p('9'|'x'): 0.0
p('a'|'x'): 0.07779316712834719
p('b'|'x'): 0.0
p('c'|'x'): 0.13896583564173592
p('d'|'x'): 0.0
p('e'|'x'): 0.06209602954755309
p('f'|'x'): 0.0
p('g'|'x'): 0.0
p('h'|'x'): 0.013619575253924284
```

```
p('i'|'x'): 0.11842105263157894
p('j'|'x'): 0.0
p('k'|'x'): 0.0
p('l'|'x'): 0.0004616805170821791
p('m'|'x'): 0.0
p('n'|'x'): 0.0
p('o'|'x'): 0.0011542012927054479
p('p'|'x'): 0.3127885503231764
p('q'|'x'): 0.0009233610341643582
p('r'|'x'): 0.0
p('s'|'x'): 0.0
p('t'|'x'): 0.10872576177285319
p('u'|'x'): 0.0039242843951985225
p('v'|'x'): 0.036011080332409975
p('w'|'x'): 0.0
p('x'|'x'): 0.03647276084949215
p('y'|'x'): 0.0009233610341643582
p('z'|'x'): 0.0


Symbol: y
Marginal Probability p('y'): 0.01446151629653677
Transition Probabilities:
p('='|'y'): 0.0
p(' '|'y'): 0.5563625990213074
p('-'|'y'): 0.0032915670053323388
p(','|'y'): 0.06885958175155252
p(';'|'y'): 0.001996883983234952
p(':'|'y'): 0.00223826556362599
p('!'|'y'): 0.005837045489456014
p('?'|'y'): 0.004783744047749666
p('/'|'y'): 2.1943780035548923e-05
p('.'|'y'): 0.04733273353667903
p('''|'y'): 0.0
p('"'|'y'): 0.0
p('('|'y'): 0.0
p(')'|'y'): 0.0008119198613153102
p('['|'y'): 0.0
p(']'|'y'): 0.0
p('*'|'y'): 0.0
p('0'|'y'): 0.0
p('1'|'y'): 0.0
p('2'|'y'): 0.0
p('3'|'y'): 0.0
p('4'|'y'): 0.0
p('5'|'y'): 0.0
p('6'|'y'): 0.0
```

```
p('7'|'y'): 0.0
p('8'|'y'): 0.0
p('9'|'y'): 0.0
p('a'|'y'): 0.02563033508152114
p('b'|'y'): 0.0032696232252967896
p('c'|'y'): 0.00366461126593667
p('d'|'y'): 0.00021943780035548925
p('e'|'y'): 0.056154133110969694
p('f'|'y'): 0.0021504904434837945
p('g'|'y'): 0.0001974940203199403
p('h'|'y'): 0.0002633253604265871
p('i'|'y'): 0.02361150731825064
p('j'|'y'): 0.0
p('k'|'y'): 0.0007899760812797612
p('l'|'y'): 0.004432643567180883
p('m'|'y'): 0.0036207237058655723
p('n'|'y'): 0.0009216387614930548
p('o'|'y'): 0.13139935485286697
p('p'|'y'): 0.001141076561848544
p('q'|'y'): 0.0
p('r'|'y'): 0.0019529964231638542
p('s'|'y'): 0.028417195146035856
p('t'|'y'): 0.017445305128261396
p('u'|'y'): 0.001163020341884093
p('v'|'y'): 2.1943780035548923e-05
p('w'|'y'): 0.001777446182879463
p('x'|'y'): 0.0
p('y'|'y'): 0.0
p('z'|'y'): 0.00021943780035548925


Symbol: z
Marginal Probability p('z'): 0.0007092556433424696
Transition Probabilities:
p('='|'z'): 0.0
p(' '|'z'): 0.0348993288590604
p('-'|'z'): 0.0022371364653243847
p(','|'z'): 0.013870246085011185
p(';'|'z'): 0.00044742729306487697
p(':'|'z'): 0.00044742729306487697
p('!'|'z'): 0.0013422818791946308
p('?'|'z'): 0.0008948545861297539
p('/'|'z'): 0.0
p('.'|'z'): 0.006263982102908278
p('''|'z'): 0.0
p('"'|'z'): 0.0
p('('|'z'): 0.0
```

```
p(')'|'z'): 0.00044742729306487697
p('['|'z'): 0.0
p(']'|'z'): 0.0
p('*'|'z'): 0.0
p('0'|'z'): 0.0
p('1'|'z'): 0.0
p('2'|'z'): 0.0
p('3'|'z'): 0.0
p('4'|'z'): 0.0
p('5'|'z'): 0.0
p('6'|'z'): 0.0
p('7'|'z'): 0.0
p('8'|'z'): 0.0
p('9'|'z'): 0.0
p('a'|'z'): 0.04429530201342282
p('b'|'z'): 0.0
p('c'|'z'): 0.0
p('d'|'z'): 0.012080536912751677
p('e'|'z'): 0.3261744966442953
p('f'|'z'): 0.0
p('g'|'z'): 0.0
p('h'|'z'): 0.06890380313199106
p('i'|'z'): 0.10782997762863535
p('j'|'z'): 0.0
p('k'|'z'): 0.0017897091722595079
p('l'|'z'): 0.019686800894854587
p('m'|'z'): 0.028187919463087248
p('n'|'z'): 0.007606263982102908
p('o'|'z'): 0.28232662192393737
p('p'|'z'): 0.0
p('q'|'z'): 0.0
p('r'|'z'): 0.0
p('s'|'z'): 0.0008948545861297539
p('t'|'z'): 0.0
p('u'|'z'): 0.011633109619686801
p('v'|'z'): 0.0
p('w'|'z'): 0.00044742729306487697
p('x'|'z'): 0.0
p('y'|'z'): 0.006263982102908278
p('z'|'z'): 0.021029082774049218
```