

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)	3
1.1	Part A: Why MVG would not be an appropriate model?	3
1.2	Part B: Maximum likelihood (ML) estimate of \mathbf{p}	4
1.3	Part C: Maximum a posteriori (MAP) estimate for \mathbf{p}	6
1.4	Part D: Learning the maximum likelihood parameters	8
1.5	Part E: Learning the maximum a posteriori parameters	10
2	Model Selection (15 marks)	12
2.1	Part A: Bernoulli distributions with $p_d = 0.5$	13
2.2	Part B: Bernoulli distributions with $p_d = k$	14
2.3	Part C: Individual Bernoulli distributions	16
3	EM for Binary Data (65 marks + 5 bonus)	18
3.1	Part A: Likelihood for a mixture of K multivariate Bernoulli	18
3.2	Part B: Responsibility of our mixture components on our data	19
3.3	Part C: Maximizing parameters for the expected log-joint	20
3.4	Part D: Implementing the EM Algorithm	23
3.5	Part E: Comments on part D and plot the P matrices	29
4	LGSSMs, EM and SSID (35 bonus)	31
4.1	Part A: Make plots using the code provided	31
5	Decrypting Messages with MCMC (70 marks)	36
5.1	Part A: Learn the ML parameters for the marginal and transition probabilities	36
5.2	Part B: Write down the joint probability of the encrypted text	40
5.3	Part C: Metropolis-Hastings Algorithm	41

5.4	Part D: Implement the Metropolis-Hastings Algorithm	42
5.5	Part E: Ergodicity of the Chain	55
5.6	Part F: Further Qualitative Analysis	56
6	Implementing Gibbs sampling for LDA (60 bonus)	57
7	Optimization (15 marks)	58
7.1	Part A: Find Extrema	58
7.2	Part B: Newton's Method	60
8	Eigenvalues as solutions of an optimization problem (20 bonus)	61
8.1	Part A: Show $\sup_{x \in \mathbb{R}^n} R_A(x) = \sup_{\hat{x} \in S} R_A(\hat{x})$	61
8.2	Part B: Show that $R_A(x) \leq \lambda_1$	62
8.3	Part C: x not in $\text{span}\{\xi_1, \dots, \xi_k\}$, implies $R_A(x) < \lambda_1$	63

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 $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}$
- $\mathbf{x}^{(i)} \in \{0, 1\}^D$, $D = \text{rows} \times \text{cols. of image}$
- $\mathbf{x}^{(i)} = (x^{(1)}, \dots, x^{(D)})$

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 \mathbf{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.

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

\hat{p}_{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 ...

$$\begin{aligned} \nabla_{\mathbf{p}} f(S \mid \mathbf{p}) &= \nabla_{\mathbf{p}} \left(\prod_{i=1}^N f(\mathbf{x}^{(i)} \mid \mathbf{p}) \right) \\ &= \nabla_{\mathbf{p}} \left(\prod_{i=1}^N \prod_{d=1}^D p_d^{x_d^{(i)}} (1 - p_d)^{(1-x_d^{(i)})} \right) = 0 \end{aligned}$$

This is the equation we must solve to find \hat{p}_{MLE} .

We can also instead solve $\nabla_{\mathbf{p}} \log(f) = 0$ since the logarithm is monotonic.

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

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 $j \neq d$:

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

$j = d$:

$$\begin{aligned} \frac{\partial}{\partial p_d}(\dots) &= \frac{x_d^{(i)}}{p_d} - \frac{(1 - x_d^{(i)})}{(1 - p_d)} \\ \nabla_{\mathbf{p}} \sum_{i=1}^N \sum_{d=1}^D &\left(x_d^{(i)} \log(p_d) + (1 - x_d^{(i)}) \log(1 - p_d) \right) \end{aligned}$$

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

$$\begin{aligned} \sum_{i=1}^N \left(\frac{x_d^{(i)}}{p_d} - \frac{(1 - x_d^{(i)})}{(1 - p_d)} \right) &= 0 \\ \Rightarrow \sum_{i=1}^N \frac{(1 - p_d)x_d^{(i)} - p_d(1 - x_d^{(i)})}{p_d(1 - p_d)} & \\ = (p_d(1 - p_d))^{-1} \sum_{i=1}^N (x_d^{(i)} - x_d^{(i)} p_d - p_d + x_d^{(i)} p_d) & \\ \Rightarrow (p_d(1 - p_d))^{-1} \sum_{i=1}^N (x_d^{(i)} - p_d) &= 0 \\ \therefore (p_d(1 - p_d))^{-1} \sum_{i=1}^N x_d^{(i)} = (p_d(1 - p_d))^{-1} N p_d & \\ \therefore p_d = \frac{1}{N} \sum_{i=1}^N x_d^{(i)} & \\ \therefore \hat{p}_{\text{ML}} = (p_1, \dots, p_D) & \end{aligned}$$

1.3 Part C: Maximum a posteriori (MAP) estimate for \mathbf{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}_{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}_{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.

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

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

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

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

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

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

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

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

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

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...

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

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

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

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

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

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

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...

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

We will use...

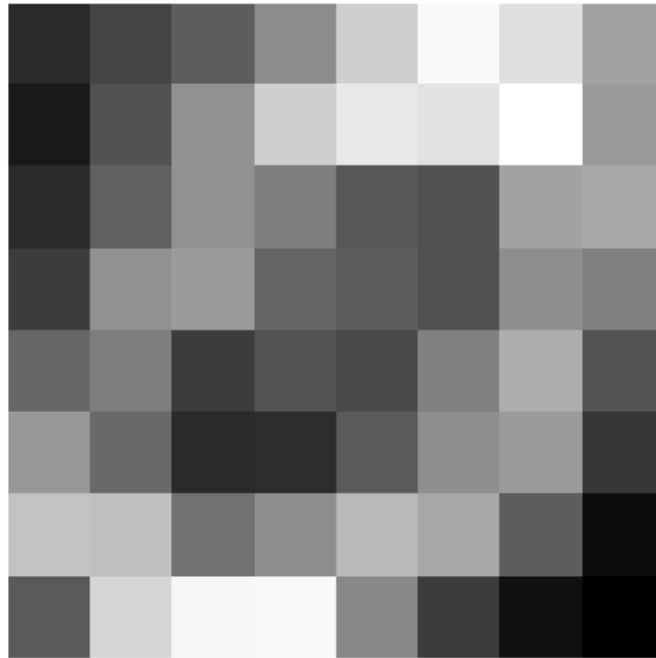
$$\hat{p}_{\text{ML}} = \frac{1}{N} X^\top \mathbf{1}_N.$$

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

```
1 import numpy as np
2 from matplotlib import pyplot as plt
3
4 # load the data set
5 X = np.loadtxt('binarydigits.txt')
6
7 def p_mle(X):
8     """
9     Computes the MLE estimate vector for a matrix X.
10    """
11    N = X.shape[0] # Number of samples (rows)
12    return np.sum(X, axis=0) / N # Equivalent to (1/N) * X.T @ ones(N)
13
14 p_mle_estimate = p_mle(X)
15
16 plt.figure()
17 plt.imshow(np.reshape(p_mle_estimate, (8,8)),
18            interpolation="None",
19            cmap='gray')
20 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...

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

we will use...

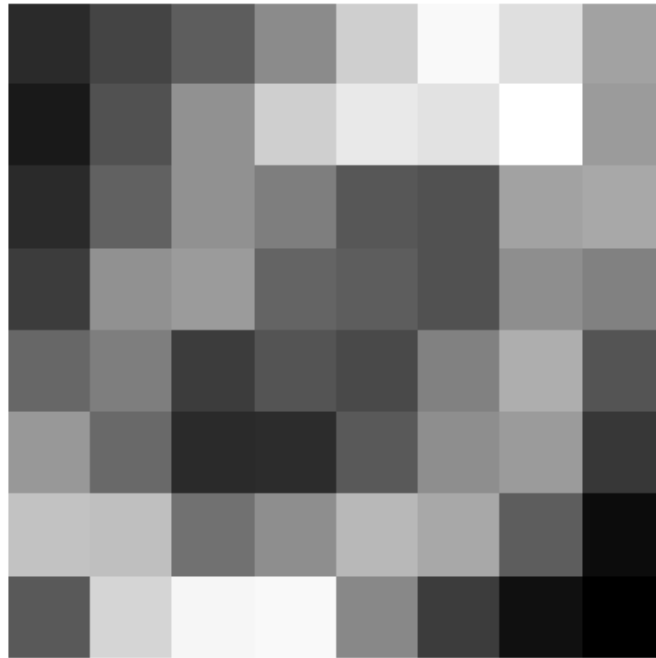
$$\hat{p}_{\text{MAP}} = \frac{\alpha - 1 + X^T \mathbf{1}_N}{\alpha + \beta + N - 2}$$

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

```
1 import numpy as np
2 from matplotlib import pyplot as plt
3
4 # load the data set
5 X = np.loadtxt('binarydigits.txt')
6
7 def p_map(X, alpha, beta):
8     """Computes the maximum a posteriori estimate of the parameter vector for
9     a matrix X. Where X is a matrix of binary digits, we assume all pixels
10     are being sampled from a multivariate Bernoulli distribution.
11     """
12     N = X.shape[0] # Number of samples (rows)
13
14     numerator = alpha - 1 + np.sum(X, axis=0)
15     denominator = alpha + beta + N - 2
16     return numerator / denominator
17
18 p_map_estimate = p_map(X, alpha=3, beta=3)
19
20 # Plot the MAP estimate
21 plt.figure()
22 plt.imshow(
23     np.reshape(p_map_estimate, (8,8)),
24     interpolation="None",
25     cmap='gray',
26 )
27 plt.axis('off')
28 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...

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

Where $S = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\}$, 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 brevity, 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...

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

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 $p_d = \frac{1}{2} \dots$

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

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

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

Code:

```
1 import numpy as np
2
3 # load the data set
4 X = np.loadtxt('binarydigits.txt')
5
6 N, D = X.shape
7
8 # Likelihood when p_d = 0.5
9 log_likelihood = N*D*np.log(1/2)
10 log_prior = np.log(1/3)
11 log_posterior = log_likelihood + log_prior
12
13 print(f'Log-Posterior when p_d = 0.5: {log_posterior:.2f}')
```

Result:

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

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

Mathematical Simplification:

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

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

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

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

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

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

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

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 $\Gamma(n) = (n-1)!$...

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

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...

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

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

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

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

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

Code:

```

1 import numpy as np
2 from math import lgamma
3
4 # load the data set
5 X = np.loadtxt('binarydigits.txt')
6
7 # Calculate the count of 1s and 0s in the data set
8 count_1 = np.count_nonzero(X == 1)
9 count_0 = np.count_nonzero(X == 0)
10
11 # Due to numerical instability, we calculate the log of the beta function
12 # using the log-gamma function...
13 # log(B(m, n)) = log(gamma(m)) + log(gamma(n)) - log(gamma(m + n))
14 log_beta = lambda m, n : lgamma(m) + lgamma(n) - lgamma(m + n)
15
16 log_likelihood = log_beta(count_1 + 1, count_0 + 1)
17 log_prior = np.log(1/3)
18 log_posterior = log_likelihood + log_prior
19
20 print(f'Log-Posterior when p_d = k: {log_posterior:.2f}')
```

Result:

Log-Posterior when $p_d = k$: -4284.82 ($\implies 0 < \text{Posterior} \ll 1$)

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...

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

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

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

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

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

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

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

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

Code:

```
1 import numpy as np
2 from math import lgamma
3
4 # load the data set
5 X = np.loadtxt('binarydigits.txt')
6
7 N, D = X.shape
8
9 # log(B(m, n)) = log(gamma(m)) + log(gamma(n)) - log(gamma(m + n))
10 log_beta = lambda m, n : lgamma(m) + lgamma(n) - lgamma(m + n)
11
12 # Calculate all the log(Beta())
13 logB_sum = 0
14 for d in range(D):
15     # Calculate the count of 1s and 0s in the data set
16     count_1_d = np.count_nonzero(X[:, d] == 1)
17     count_0_d = np.count_nonzero(X[:, d] == 0)
18
19     logB_sum += log_beta(count_1_d + 1, count_0_d + 1)
20
21 log_likelihood = logB_sum
22 log_prior = np.log(1/3)
23 log_posterior = log_likelihood + log_prior
24
25 print(f'Log-Posterior with individual Bernoullis: {log_posterior:.2f}')
```

Result:

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

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 $\pi = (\pi_1, \dots, \pi_K)$ 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

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

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 $\{1, \dots, K\}$, 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...

$$\begin{aligned} & \left\langle \sum_n \log P(\mathbf{x}^{(n)}, s^{(n)} \mid \pi, \mathbf{P}) \right\rangle_{q(\{s^{(n)}\})} \\ &= \sum_{k=1}^K \sum_{n=1}^N \log P(\mathbf{x}^{(n)}, s^{(n)} = k \mid \pi, \mathbf{P}) \cdot q(s^{(n)} = k) \\ &= \sum_{k=1}^K \sum_{n=1}^N \log (P(\mathbf{x}^{(n)} \mid s^{(n)} = k, \pi, \mathbf{P}) \cdot P(s^{(n)} = k \mid \pi, \mathbf{P})) \cdot q(s^{(n)} = k) \\ &= \sum_{k=1}^K \sum_{n=1}^N \log \left(\left(\prod_{d=1}^D p_{kd}^{x_d^{(n)}} (1 - p_{kd})^{(1-x_d^{(n)})} \right) \cdot P(s^{(n)} = k \mid \pi, \mathbf{P}) \right) \cdot q(s^{(n)} = k) \\ &= \sum_{k=1}^K \sum_{n=1}^N \log \left(\left(\prod_{d=1}^D p_{kd}^{x_d^{(n)}} (1 - p_{kd})^{(1-x_d^{(n)})} \right) \cdot \pi_k \right) \cdot q(s^{(n)} = k) \\ &= \sum_{k=1}^K \sum_{n=1}^N \log \left(\left(\prod_{d=1}^D p_{kd}^{x_d^{(n)}} (1 - p_{kd})^{(1-x_d^{(n)})} \right) \cdot \pi_k \right) \cdot r_{nk} \\ &= \sum_{k=1}^K \sum_{n=1}^N \left(\sum_{d=1}^D \left(x_d^{(n)} \log(p_{kd}) + (1 - x_d^{(n)}) \log(1 - p_{kd}) \right) + \log \pi_k \right) \cdot r_{nk} \end{aligned}$$

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...

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

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

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

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

and K equations of the form...

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

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} ...

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

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

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

Applying this formula, we get that...

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

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

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

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

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 $\lambda \neq 0$) ...

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

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

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...

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

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

Helper Functions:

```
1 import numpy as np
2 from matplotlib import pyplot as plt
3 from pprint import pprint
4
5 # load the data set
6 X = np.loadtxt('binarydigits.txt')
7
8 np.random.seed(1)
9
10 def E_step(X, pi, P):
11     """Perform the E-step of the EM algorithm for a Multivariate Bernoulli
12     ↪ Mixture Model.
13
14     Input:
15     X: N x D matrix of binary data
16     pi: K x 1 vector of mixing coefficients
17     P: K x D matrix of Bernoulli parameters
18
19     Output:
20     r: N x K matrix of responsibilities
21     """
22     N, D = X.shape
23     K = len(pi)
24     r = np.zeros((N, K))
25     for n in range(N):
26         denominator = np.sum([pi[j] * np.prod((P[j, :] ** X[n, :]) * ((1 -
27     ↪ P[j, :]) ** (1 - X[n, :])) for j in range(K)])
28         for k in range(K):
29             numerator = pi[k] * np.prod((P[k, :] ** X[n, :]) * ((1 - P[k, :])
30     ↪ ** (1 - X[n, :]))
31             r[n, k] = numerator / denominator
32     return r
33
34 def M_step(X, r):
35     """Perform the M-step of the EM algorithm for a Multivariate Bernoulli
36     ↪ Mixture Model.
37
38     Input:
39     X: N x D matrix of binary data
40     r: N x K matrix of responsibilities
41
42     Output:
43     pi: K x 1 vector of mixing coefficients
44     P: K x D matrix of Bernoulli parameters
```

```

42     """
43     N, D = X.shape
44     K = r.shape[1]
45     pi = np.sum(r, axis=0) / N
46     P = np.zeros((K, D))
47     for k in range(K):
48         for d in range(D):
49             P[k, d] = np.sum(X[:, d] * r[:, k]) / np.sum(r[:, k])
50     return pi, P
51
52
53 def log_likelihood(X, pi, P):
54     """Calculate the log-likelihood of the data under the current model.
55
56     Input:
57     X: N x D matrix of binary data
58     pi: K x 1 vector of mixing coefficients
59     P: K x D matrix of Bernoulli parameters
60
61     Output:
62     log_likelihood: log-likelihood of the data
63     """
64     N, D = X.shape
65     K = len(pi)
66     log_likelihood = 0
67
68     # Calculate the log-likelihood
69     for n in range(N):
70         likelihood_for_1_image = 0
71         for k in range(K):
72             likelihood_for_1_image += pi[k] * np.prod((P[k, :]**X[n,
73                 ↪      :])*((1-P[k, :])**(1-X[n, :])))
74         log_likelihood += np.log(likelihood_for_1_image)
75     return log_likelihood

```

EM Algorithm:

```
1 def em(X, K, epsilon=1e-16, max_iter=1000):
2     """We implement the EM algorithm for a Multivariate Bernoulli Mixture
3     Model. We are trying to find the pi, and P matrix (K x D) that
4     maximizes the log-likelihood of the data X, under some
5     distribution q of the mixture model components.
6
7     Input:
8     X: N x D matrix of binary data
9     K: number of components
10    epsilon: minimum change in log-likelihood to continue the iterations
11    max_iter: maximum number of iterations
12
13    Output:
14    pi: K x 1 vector of mixing coefficients
15    P: K x D matrix of Bernoulli parameters
16    log_likelihoods: list of log-likelihoods at each iteration
17    """
18    # Initialize the parameters
19    N, D = X.shape
20    log_likelihoods = []
21    delta = np.inf
22    i = 0
23
24    # pi and P randomly sampled from a uniform distribution U(0, 1)
25    # and appropriately normalized
26    pi, P = np.random.rand(K), np.random.rand(K, D)
27    pi = pi / pi.sum()
28
29    while delta > epsilon and i < max_iter:
30        # Calculate the responsibilities
31        r = E_step(X, pi, P)
32
33        # Calculate the parameters
34        pi, P = M_step(X, r)
35
36        # Calculate the log-likelihood
37        log_likelihoods.append(log_likelihood(X, pi, P))
38
39        # Check for convergence
40        if i > 0: delta = np.abs(log_likelihoods[i] - log_likelihoods[i - 1])
41        i += 1
42    return pi, P, log_likelihoods
```

Running The EM Algorithm:

```
1 K = [2, 3, 4, 7, 10]
2 K_log_likelihoods = []
3 for i in range(len(K)):
4     # Run the EM algorithm
5     pi, P, log_likelihoods = em(X, K[i])
6     K_log_likelihoods.append(log_likelihoods)
7
8     # Print the results
9     print(f'K = {K[i]}, Iterations = {len(log_likelihoods)},')
10    print(f'Final-Log-likelihood = {log_likelihoods[-1]}, Pi =')
11    pprint(pi)
12    print()
13
14    # Plot the Bernoulli parameters in one figure
15    fig, axs = plt.subplots(1, K[i], figsize=(10*K[i], int(5*np.cbrt(K[i]))))
16    for j in range(len(P)):
17        axs[j].imshow(P[j].reshape(8, 8), cmap='gray', interpolation='none')
18        axs[j].set_title(f'Component {j+1}')
19        axs[j].axis('off')
20    plt.savefig(f'q3/bernoulli_parameters_K_{K[i]}.png')
21
22    # Plot the log-likelihood for all K
23    plt.figure()
24    plt.title('Log-likelihood for all K')
25    plt.grid()
26    colors = ['b', 'g', 'r', 'c', 'm']
27    for i in range(len(K)):
28        l = len(K_log_likelihoods[i])
29        plt.plot(range(1, l+1), K_log_likelihoods[i], label=f'K = {K[i]}',
30                ↪ color=colors[i])
31        # Plot the Iteration value where the log-likelihood converges
32        plt.axvline(x=l, linestyle='--', color=colors[i])
33    plt.xlabel('Iteration')
34    plt.ylabel('Log-likelihood')
35    plt.legend()
36    plt.savefig('q3/log_likelihood_all_K.png')
```

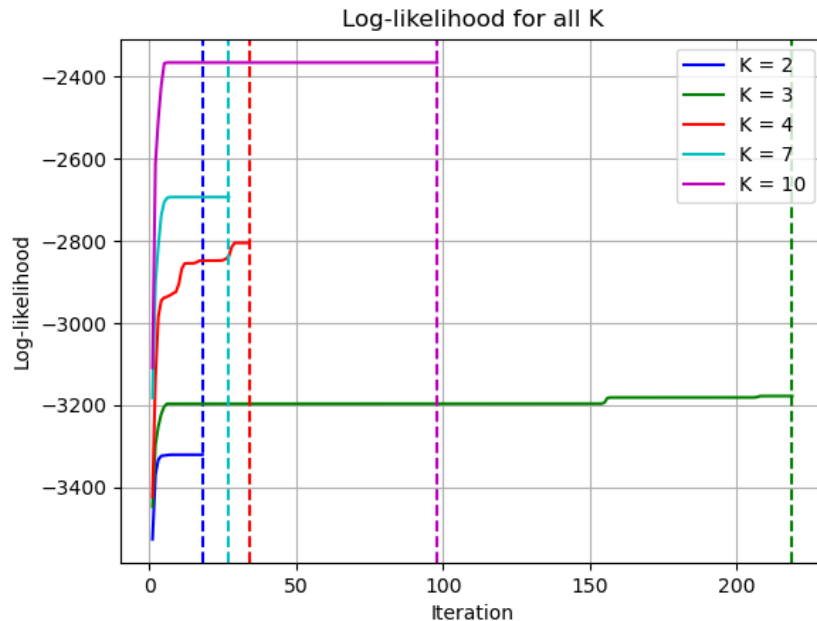
Results from Code:

```

1 K = 2, Iterations = 18,
2 Final-Log-likelihood = -3320.3775169778814, Pi =
3 array([0.65962235, 0.34037765])
4
5 K = 3, Iterations = 219,
6 Final-Log-likelihood = -3177.5571427467785, Pi =
7 array([0.24989042, 0.15999792, 0.59011166])
8
9 K = 4, Iterations = 34,
10 Final-Log-likelihood = -2804.826323213882, Pi =
11 array([0.17          , 0.25000012, 0.39999988, 0.18          ])
12
13 K = 7, Iterations = 27,
14 Final-Log-likelihood = -2693.2531832001287, Pi =
15 array([0.05999988, 0.29000169, 0.02          , 0.21930991, 0.04          ,
16         0.14068857, 0.22999995])
17
18 K = 10, Iterations = 98,
19 Final-Log-likelihood = -2365.828343399904, Pi =
20 array([0.04          , 0.16          , 0.09177852, 0.15074369, 0.03          ,
21         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: $K = 2$

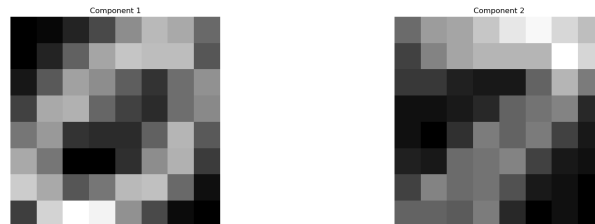


Figure 5: $K = 3$

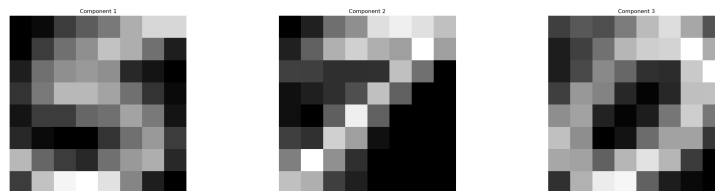


Figure 6: $K = 4$

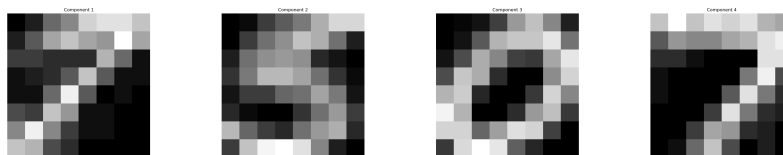


Figure 7: $K = 7$

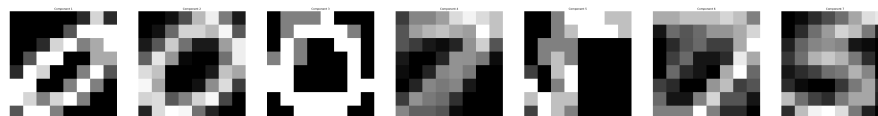


Figure 8: $K = 10$



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:

```
1 np.random.seed(0)
2
3 ssm_train_path = 'q4_ssm/ssm_spins.txt'
4 ssm_train = []
5 with open(ssm_train_path, 'r') as f:
6     ssm_train = f.readlines()
7     ssm_train = np.array([list(map(float, x.strip().split())) for x in
8         ↪ ssm_train])
9
10 # SSM parameters and variables predefined in the problem sheet
11 y_init = np.random.randn(4)
12
13 # shorthand for the matrix A
14 s_180 = np.sin(2*np.pi/180)
15 s_90 = np.sin(2*np.pi/90)
16 c_180 = np.cos(2*np.pi/180)
17 c_90 = np.cos(2*np.pi/90)
18
19 A = 0.99 * np.array([
20     [c_180, -s_180, 0, 0],
21     [s_180, c_180, 0, 0],
22     [0, 0, c_90, -s_90],
23     [0, 0, s_90, c_90]
24 ])
25
26 Q = np.eye(4) - A.dot(A.T)
27
28 R = np.eye(5)
29
30 C = np.array([
31     [1, 0, 1, 0],
32     [0, 1, 0, 1],
33     [1, 0, 0, 1],
34     [0, 0, 1, 1],
35     [0.5, 0.5, 0.5, 0.5]
36 ])
```



```

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]')

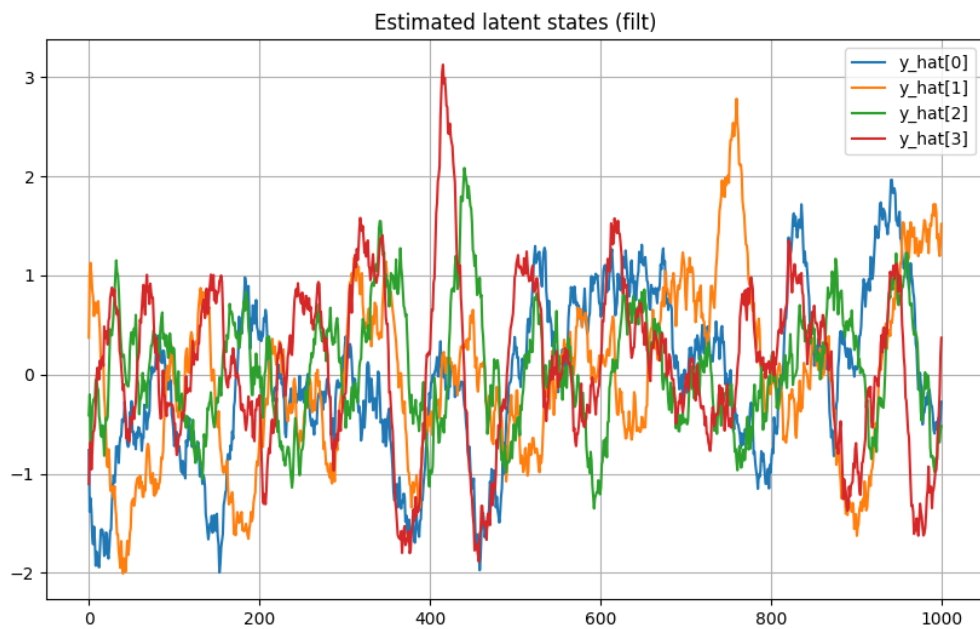
```

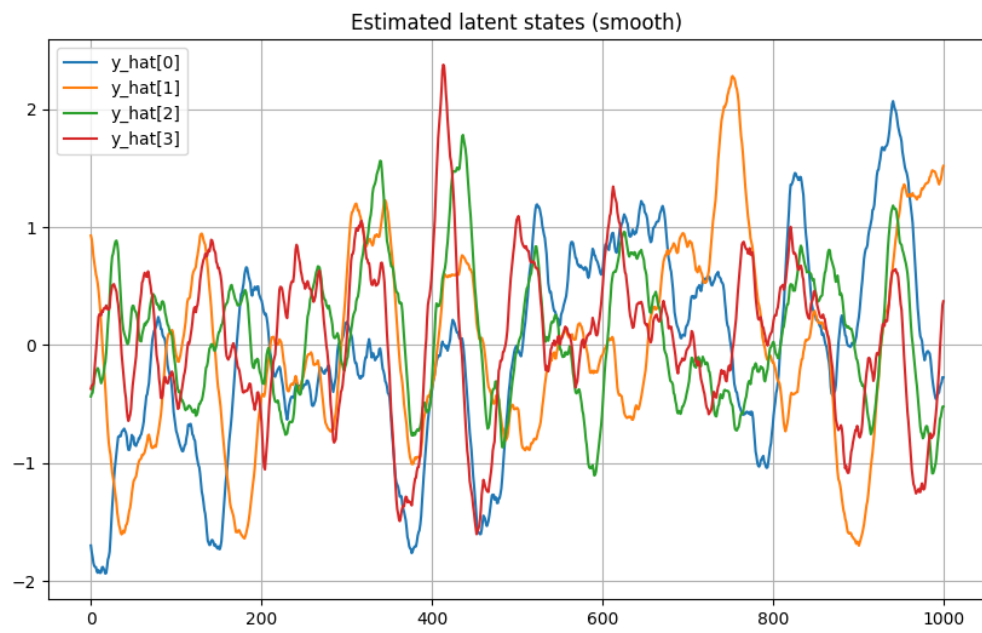
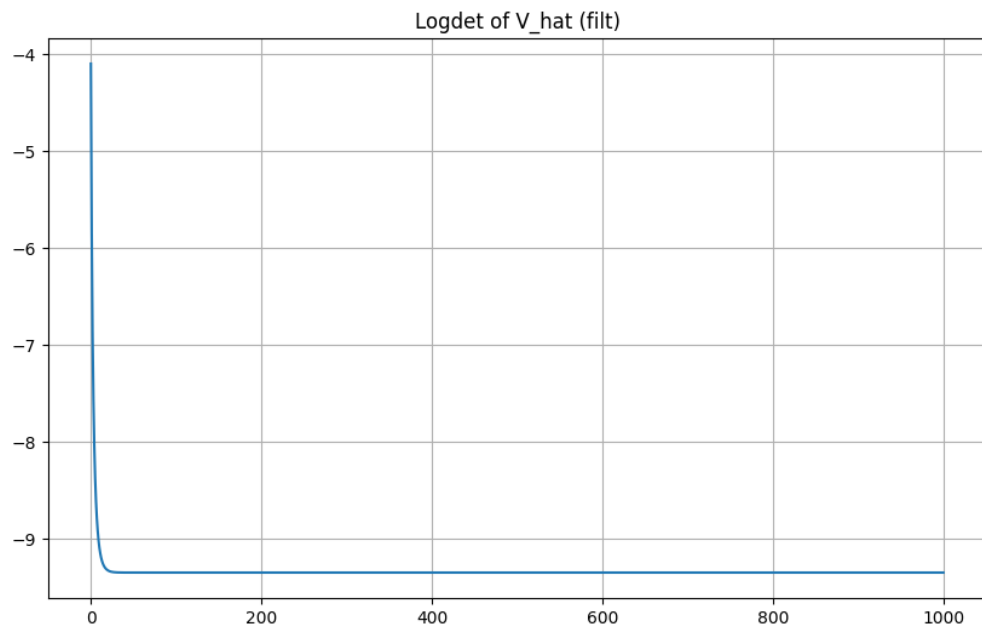
```

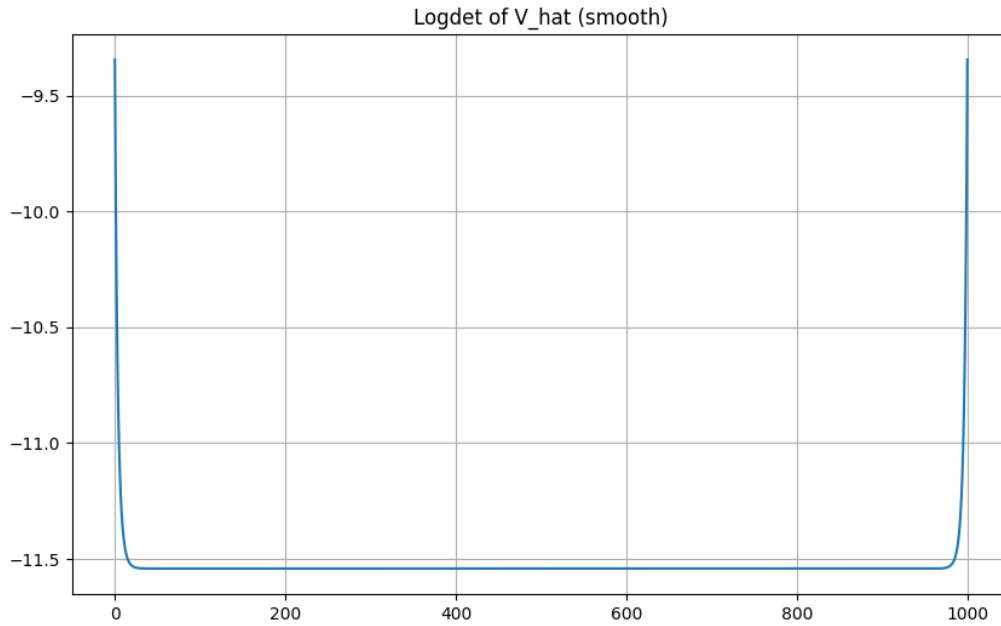
83 plt.plot(y_hat[2], label='y_hat[2]')
84 plt.plot(y_hat[3], label='y_hat[3]')
85 plt.legend()
86 plt.grid()
87 plt.title('Estimated latent states (smooth)')
88 plt.savefig('q4_ssm/estimated_latent_states_smooth.png')
89
90 # Plot the logdet of v_hat
91 plt.figure(figsize=(10, 6))
92 plt.plot([logdet(V_hat[i]) for i in range(V_hat.shape[0])])
93 plt.grid()
94 plt.title('Logdet of V_hat (smooth)')
95 plt.savefig('q4_ssm/logdet_v_hat_smooth.png')

```

Plots:







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 $1 \rightarrow 1$, where $a \rightarrow b$, and $a \not\rightarrow (b, c)$.
- ‘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, \dots, S_n under a Markov chain, $p(S_1, S_2, \dots, S_n) = p(S_1) \prod_{i=2}^n p(S_i) \cdot p(S_i \mid S_{i-1})$.

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 = Number of Symbols in “War and Peace”,

$\text{Count}(s)$ = Count the number of times s 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:

```
1 import numpy as np
2 from collections import defaultdict
3 import matplotlib.pyplot as plt
4
5 # Load the text file and convert it to a numpy array of characters
6 wap_path = "war_and_peace.txt"
7 war_and_peace = ""
8 with open(wap_path, 'r', encoding='utf-8') as file:
9     war_and_peace = file.read()
10    # Clean the text, removing newlines and make it lowercase
11    war_and_peace = war_and_peace.replace("\n", " ").replace("\r",
    ↪    ").lower()
12 wap_np = np.array(list(war_and_peace))
13
14
15 # Load the list of symbols and convert it to a numpy array
16 symbols_file_path = "symbols.txt"
17 symbols = []
18 with open(symbols_file_path, 'r', encoding='utf-8') as sym_file:
19     symbols = [line.strip() for line in sym_file.readlines()]
20    # Second symbol is the space character
21    symbols[1] = " "
22 symbols_np = np.array(symbols)
23
24
25 # Create a dictionary to store the count of each symbol and symbol pair
26 # and iterate over the text to count the occurrences of each symbol and
    ↪ symbol pair
27 symbols_count = defaultdict(int)
28 symbol_pairs_count = defaultdict(int)
29 n = len(wap_np)
30 for i in range(n - 1):
31     current_pair = (wap_np[i], wap_np[i + 1])
32     # Ensure current pair is valid (exists in symbols list)
33     if current_pair[0] in symbols and current_pair[1] in symbols:
34         symbols_count[current_pair[0]] += 1
35         symbol_pairs_count[current_pair] += 1
36 # Edge case, count the last symbol
37 if wap_np[-1] in symbols:
38     symbols_count[wap_np[-1]] += 1
39
40
41 # Compute the transition probability matrix and marginal probability vector
42 num_symbols = len(symbols)
43 transition_matrix = np.zeros((num_symbols, num_symbols))
```

```

44 marginal_probabilities = np.zeros(num_symbols)
45 # Fill in the transition matrix with counts of specific symbol pairs
46 for current_pair, count in symbol_pairs_count.items():
47     i = symbols.index(current_pair[0])
48     j = symbols.index(current_pair[1])
49     transition_matrix[i, j] = count
50 # Normalize the rows of the transition matrix to get probabilities
51 for i in range(num_symbols):
52     row_sum = sum(transition_matrix[i, :])
53     if row_sum > 0:
54         transition_matrix[i, :] /= row_sum
55 # Fill marginal probabilities vector
56 total_symbols_count = sum(symbols_count.values())
57 for symbol, count in symbols_count.items():
58     i = symbols.index(symbol)
59     marginal_probabilities[i] = count / total_symbols_count

```

Plot Our Transition and Marginal Probabilities:

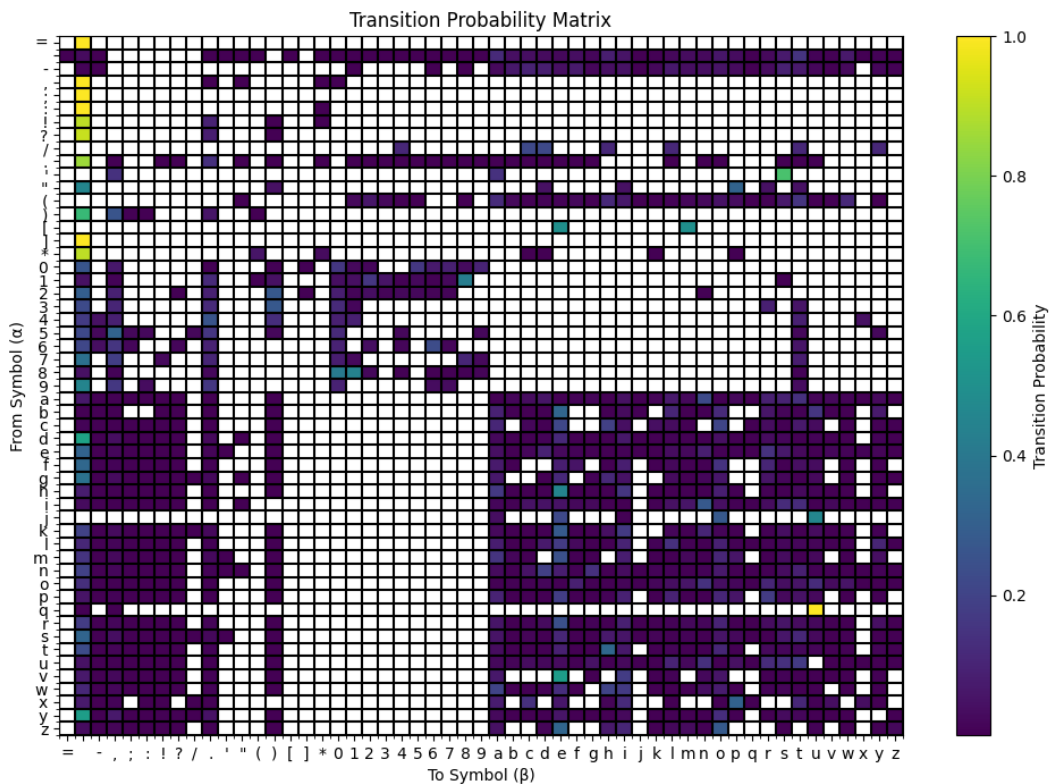
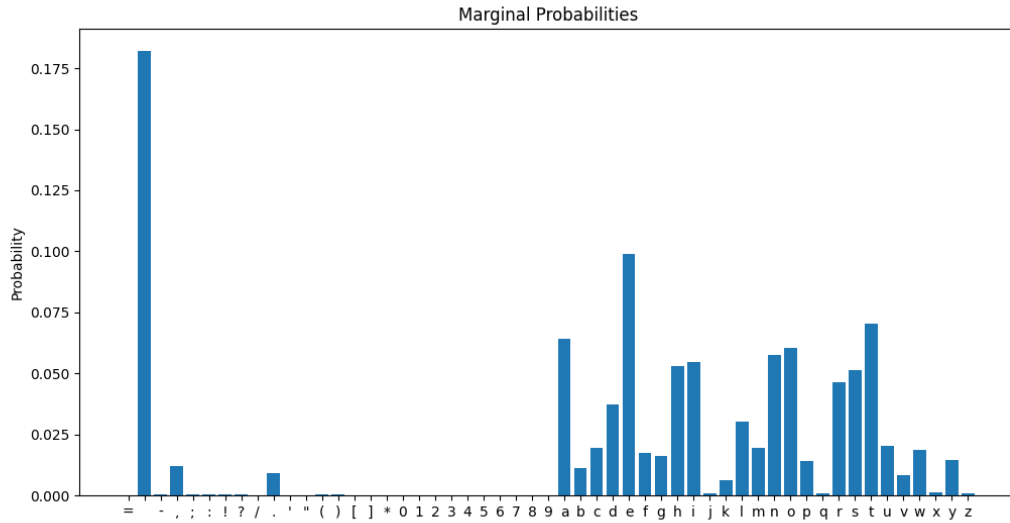
```

1 # Create a custom colormap where masked (zero) values are displayed as white
2 masked_matrix = np.ma.masked_where(transition_matrix == 0, transition_matrix)
3 cmap = plt.cm.viridis
4 cmap.set_bad(color='white')
5 # Plot the transition matrix with custom colormap and gridlines
6 plt.figure(figsize=(12, 8))
7 plt.imshow(masked_matrix, cmap=cmap, aspect='auto')
8 plt.colorbar(label="Transition Probability")
9 plt.grid(which='minor', color='black', linestyle='-', linewidth=1.5)
10 plt.xticks(ticks=np.arange(num_symbols), labels=symbols)
11 plt.yticks(ticks=np.arange(num_symbols), labels=symbols)
12 plt.gca().set_xticks(np.arange(-.5, num_symbols, 1), minor=True)
13 plt.gca().set_yticks(np.arange(-.5, num_symbols, 1), minor=True)
14 plt.title("Transition Probability Matrix")
15 plt.xlabel("To Symbol ()")
16 plt.ylabel("From Symbol ()")
17 plt.savefig("q5a_transition_prob")
18
19
20 # Plot the marginal probabilities as a bar plot
21 plt.figure(figsize=(12, 6))
22 plt.bar(np.arange(num_symbols), marginal_probabilities)
23 plt.xticks(ticks=np.arange(num_symbols), labels=symbols)
24 plt.title("Marginal Probabilities")
25 plt.ylabel("Probability")
26 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...

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

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

```

1:  $x_t \leftarrow x_0$  ▷ Initial value of chain
2:  $t \leftarrow 0$ 
3: for  $t = 1, 2, \dots$  do
4:   Generate a candidate  $x'$  from proposal distribution  $q(x | y)$ 
5:
6:   Calculate the acceptance probability  $A(x, x') = \min \left( 1, \frac{\pi(x')q(x_t|x')}{\pi(x_t)q(x'|x_t)} \right)$ 
7:   Generate  $u \sim \mathcal{U}(0, 1)$ 
8:   if  $u \leq \alpha$  then
9:     Accept  $x'$  and set  $x_{t+1} = x'$ 
10:  else
11:    Reject  $x'$  and set  $x_{t+1} = x_t$ 
12:  end if
13: end for
14: return  $(x_0, \dots, x_t)$ 
```

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, $\therefore S(\sigma \rightarrow \sigma') = S(\sigma' \rightarrow \sigma)$. This in fact simplifies our algorithm greatly, to become the simpler Metropolis algorithm. Therefore, the acceptance probability simplifies...

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

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

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

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

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:

```
1 import numpy as np
2 import collections
3
4 # Load the encrypted message and transform to numpy array
5 message_path = "message.txt"
6 message: str
7 with open(message_path, 'r', encoding='utf-8') as file:
8     message = file.read()
9 message = message.replace("\n", " ").replace("\r", "").lower()
10 message_np = np.array(list(message))
11
12
13 def decrypt_message(sigma, message=message, symbols=symbols):
14     """Decrypt the message using the given mapping sigma."""
15     decrypted_text = np.array([sigma[symbols.index(c)] for c in message])
16     return "".join(decrypted_text)
17
18
19 def log_likelihood(
20     transition_matrix,
21     marginal_probabilities,
22     sigma,
23     symbols=symbols,
24 ):
25     """Calculate the likelihood of the encrypted text given the mapping
26     sigma from the unencrypted text to the encrypted text. We have to
27     reverse the mapping to get the likelihood of the unencrypted text.
28     """
29     # Map the encrypted text to the unencrypted text using sigma
30     decrypted_text = decrypt_message(sigma)
31
32     # To prevent numerical issues, we add a small constant to the transition
33     # matrix, this acts to restore ergodicity to the chain.
34     alpha = 1e-2
35
36     # Initialize the likelihood
37     log_likelihood =
38     ↪ np.log(marginal_probabilities[symbols.index(decrypted_text[0])])
39     # print(f'Log likelihood 1st term: {log_likelihood}')
40     for i in range(1, len(decrypted_text)):
41         # Get the transition probability from the previous symbol to the
42         ↪ current symbol
43         prev_symbol = decrypted_text[i - 1]
44         current_symbol = decrypted_text[i]
```

```

43     log_likelihood +=
        ↪ np.log(transition_matrix[symbols.index(prev_symbol),
        ↪ symbols.index(current_symbol)] + alpha)
44     # print(f'Log likelihood {i+1}th term: {log_likelihood}')
45
46     return log_likelihood
47
48
49 def calculate_log_acceptance_probability(
50     sigma,
51     sigma_proposed,
52 ):
53     """Calculate the log acceptance probability for the proposed state, using
        ↪ the
54     formula: min(0, log_likelihood_proposed - log_likelihood_current).
55     """
56     # Calculate the likelihood of the proposed state
57     log_likelihood_proposed = log_likelihood(
58         transition_matrix,
59         marginal_probabilities,
60         sigma_proposed,
61     )
62     # Calculate the likelihood of the current state
63     log_likelihood_current = log_likelihood(
64         transition_matrix,
65         marginal_probabilities,
66         sigma,
67     )
68     return min(0, log_likelihood_proposed - log_likelihood_current)
69
70
71 def sigma_proposal(
72     sigma,
73 ):
74     """ Generate a new proposal for the MCMC algorithm, by swapping two
        ↪ symbols in sigma. """
75     sigma_proposed = sigma.copy()
76     i, j = np.random.choice(len(sigma), 2, replace=False)
77     sigma_proposed[i], sigma_proposed[j] = sigma_proposed[j],
        ↪ sigma_proposed[i]
78     return sigma_proposed
79
80
81 def initialize_sigma(
82     symbols: list=symbols,
83     encrypted_text=message_np
84 ):

```

```

85     """Given a message as a numpy array of characters, initialize the mapping
86     ↪ sigma. Sigma
87     is a permutation of the symbols list, which maps the symbols in the
88     ↪ encrypted text to the
89     symbols in the unencrypted text. We can use the fact that the space
90     ↪ character is the most
91     common symbol in the English language to initialize sigma. We will then
92     ↪ find the most common
93     symbol in the encrypted text and map it to the space character.
94
95     Args:
96     symbols: list of symbols
97     encrypted_text: numpy array of characters representing the encrypted
98     ↪ message
99
100     Returns:
101     sigma_init: numpy array of characters representing the initial mapping
102     """
103     # Initialize the mapping sigma
104     sigma_init = np.random.permutation(symbols)
105
106     # Find the most common symbol in the encrypted text
107     counter = collections.Counter(encrypted_text)
108     most_common_symbol = counter.most_common(1)[0][0]
109
110     # Find the index of the most common symbol in the symbols list
111     most_common_symbol_index = symbols.index(most_common_symbol)
112
113     # Find the index of the space character in the symbols list
114     # space_index = symbols.index(" ")
115
116     # Swap the most common symbol with the space character in the mapping
117     ↪ sigma
118     swap_letter = sigma_init[most_common_symbol_index]
119     sigma_init[np.where(sigma_init == " ")[0]] = swap_letter
120     sigma_init[most_common_symbol_index] = " "
121
122     # Swap the space character with the most common symbol in the mapping
123     ↪ sigma
124     # sigma_init[space_index] = most_common_symbol
125
126     return sigma_init
127
128 def run_mcmc(
129     N_iters: int,
130     symbols: list,

```

```

125     smart_sigma: bool=None,
126     encrypted_text: np.array=message_np,
127     transition_matrix: np.array=transition_matrix,
128     marginal_probabilities: np.array=marginal_probabilities
129 ):
130     """Run an MCMC algorithm for a fixed number of iterations on our
131     encrypted text. We will use the Metropolis-Hastings algorithm to
132     sample from the space of possible mappings sigma. We will use the
133     log likelihood of the encrypted text as the target distribution.
134     Each proposal though, only swaps two symbols in current sigma,
135     to produce our new 'proposal' sigma. We will accept the proposal
136     with the appropriate probability, based on the ratio of the
137     log-likelihoods of the proposed and current states. We will
138     print the decrypted text every 100 iterations, along with
139     the likelihood of the current state.
140
141     To initialize intelligently, we can use the fact that the space
142     character is the most common, and initialize the mapping sigma
143     accordingly, with the smart_sigma flag.
144
145     Inputs:
146     N_iters: int, number of iterations to run the MCMC algorithm
147     symbols: list of symbols
148     smart_sigma: bool, flag to initialize the mapping sigma intelligently
149     encrypted_text: numpy array of chars representing the encrypted message
150     transition_matrix: numpy array representing the transition matrix
151     marginal_probabilities: numpy array of the marginal probabilities
152
153     Returns:
154     sigma: numpy array of characters representing the final mapping
155     """
156     if not smart_sigma: sigma = np.random.permutation(symbols)
157     if smart_sigma: sigma = initialize_sigma()
158     N_accepted = 0
159     i = 0
160     for i in range(N_iters):
161         # Initialize the acceptance flag and generate a proposal
162         accepted, sigma_proposed = False, sigma_prpoosal(sigma)
163         # Accept the proposal with the appropriate probability
164         log_acceptance_probability = calculate_log_acceptance_probability(
165             sigma,
166             sigma_proposed,
167         )
168         if np.log(np.random.rand()) < log_acceptance_probability:
169             sigma = sigma_proposed
170             N_accepted += 1
171             accepted = True

```

```

172     # Print the decrypted text every 100 iterations
173     if i % 100 == 0:
174         decrypted_text = decrypt_message(sigma)
175         decrypted_text_str = "".join(decrypted_text)
176         print(f'Decrypted Text (Iter = {i}):
177             ↳ "{decrypted_text_str[:60]}"')
178
179     # Likelihood of the current state
180     likelihood_current = log_likelihood(
181         transition_matrix,
182         marginal_probabilities,
183         sigma,
184     )
185     print(f'Likelihood (Iter = {i}): {likelihood_current:.2e}')
186     return sigma
187
188 # Run the MCMC algorithm
189 N_iters = 10000
190 sigma = run_mcmc(N_iters, symbols, smart_sigma=True)
191
192 # Decrypt the message using the final mapping sigma
193 decrypted_message = decrypt_message(sigma)
194 print(f'FINAL DECRYPTED MESSAGE: {decrypted_message}')

```

Results:

```

1 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*"
2 Likelihood (Iter = 0): -1.38e+04
3 Decrypted Text (Iter = 100): "a, nf fld,.p" c,2 n1"p mdo,p"chop fpc"i nf
  ↳ 8ctsp" .cmp np il"
4 Likelihood (Iter = 100): -8.38e+03
5 Decrypted Text (Iter = 200): "a, 1r rnd,p.h c,2 lnh. mdo,.hcbo. r.chi 1r
  ↳ !cts.h pcm. 1. in"
6 Likelihood (Iter = 200): -7.73e+03
7 Decrypted Text (Iter = 300): "s- mr rnd-p.a c-2 mna. 1do-.acbo. r.cai mr
  ↳ !cth.a pc1. m. in"
8 Likelihood (Iter = 300): -7.29e+03
9 Decrypted Text (Iter = 400): "s- kr rnd-m.a c-2 kna. 1do-.acfo. r.cai kr
  ↳ ycth.a mc1. k. in"
10 Likelihood (Iter = 400): -6.94e+03
11 Decrypted Text (Iter = 500): "s- km mld-r.a c-f kla. 1do-.ac2o. m.cai km
  ↳ ycth.a rc1. k. il"
12 Likelihood (Iter = 500): -6.79e+03
13 Decrypted Text (Iter = 600): "s- km mld-era n-f klar wdo-ranpor mrnai km
  ↳ ynthra enwr kr il"

```



```

14 Likelihood (Iter = 600): -5.87e+03
15 Decrypted Text (Iter = 700): "sk fm mldkira nkw flar -dokranpor mrnae fm
   ↳ ynthra in-r fr el"
16 Likelihood (Iter = 700): -5.54e+03
17 Decrypted Text (Iter = 800): "sk cm mlykira nkf clar uyokranpor mrnae cm
   ↳ wnthra inur cr el"
18 Likelihood (Iter = 800): -5.41e+03
19 Decrypted Text (Iter = 900): "sk cm mlikyra nkf clar uiokranpor mrnae cm
   ↳ wnthra ynur cr el"
20 Likelihood (Iter = 900): -5.37e+03
21 Decrypted Text (Iter = 1000): "sk cm mlikyra nkf clar uiokran-or mrnae cm
   ↳ wnthra ynur cr el"
22 Likelihood (Iter = 1000): -5.29e+03
23 Decrypted Text (Iter = 1100): "sm ck klimfra nmy clar uiomran-or krnae ck
   ↳ wnthra fnur cr el"
24 Likelihood (Iter = 1100): -5.22e+03
25 Decrypted Text (Iter = 1200): "sm ck klnmfra imy clar unomrai-or kriae ck
   ↳ withra fiur cr el"
26 Likelihood (Iter = 1200): -5.09e+03
27 Decrypted Text (Iter = 1300): "sa ck klnafrm iay clmr unoarmi-or krime ck
   ↳ githrm fiur cr el"
28 Likelihood (Iter = 1300): -5.02e+03
29 Decrypted Text (Iter = 1400): "sa ck konamrf iay cofr unlarfi-lr krife ck
   ↳ githrf miur cr eo"
30 Likelihood (Iter = 1400): -4.96e+03
31 Decrypted Text (Iter = 1500): "so ck kanomrf ioy cafr unlorfivlr krife ck
   ↳ githrf miur cr ea"
32 Likelihood (Iter = 1500): -4.93e+03
33 Decrypted Text (Iter = 1600): "uo cy yinomef aov cife snloefakle yeafr cy
   ↳ gathef mase ce ri"
34 Likelihood (Iter = 1600): -4.73e+03
35 Decrypted Text (Iter = 1700): "ur cy yinrkef arv cife snlrefamle yeafo cy
   ↳ gathef kase ce oi"
36 Likelihood (Iter = 1700): -4.62e+03
37 Decrypted Text (Iter = 1800): "us cy yinskef asg cife rnlsefable yeafo cy
   ↳ vathef kare ce oi"
38 Likelihood (Iter = 1800): -4.56e+03
39 Decrypted Text (Iter = 1900): "us fy yinskec asg fice rnlsecable yeaco fy
   ↳ vathec kare fe oi"
40 Likelihood (Iter = 1900): -4.55e+03
41 Decrypted Text (Iter = 2000): "us fy yinskec asg fice rnlsecable yeaco fy
   ↳ vathec kare fe oi"
42 Likelihood (Iter = 2000): -4.55e+03
43 Decrypted Text (Iter = 2100): "us fy yinskec asg fice rnlsecable yeaco fy
   ↳ vathec kare fe oi"
44 Likelihood (Iter = 2100): -4.55e+03

```

```

45 Decrypted Text (Iter = 2200): "us fy yiosked asg fide rolsedable yeadn fy
   ↪ vathed kare fe ni"
46 Likelihood (Iter = 2200): -4.42e+03
47 Decrypted Text (Iter = 2300): "us fy yiosked asl fide gorsedabre yeadn fy
   ↪ mathed kage fe ni"
48 Likelihood (Iter = 2300): -4.36e+03
49 Decrypted Text (Iter = 2400): "us fy yiosked asl fide gorsedabre yeadn fy
   ↪ mathed kage fe ni"
50 Likelihood (Iter = 2400): -4.34e+03
51 Decrypted Text (Iter = 2500): "us fy yiosked asl fide gorsedabre yeadn fy
   ↪ mathed kage fe ni"
52 Likelihood (Iter = 2500): -4.34e+03
53 Decrypted Text (Iter = 2600): "us fy yiosked asl fide gorsedabre yeadn fy
   ↪ mathed kage fe ni"
54 Likelihood (Iter = 2600): -4.34e+03
55 Decrypted Text (Iter = 2700): "us by yioskem asl bime gorsemefre yeamn by
   ↪ dathem kage be ni"
56 Likelihood (Iter = 2700): -4.34e+03
57 Decrypted Text (Iter = 2800): "un by yionkem anl bime gornemefre yeams by
   ↪ dathem kage be si"
58 Likelihood (Iter = 2800): -4.21e+03
59 Decrypted Text (Iter = 2900): "un by yiongem anl bime kornemefre yeams by
   ↪ dathem gake be si"
60 Likelihood (Iter = 2900): -4.19e+03
61 Decrypted Text (Iter = 3000): "un by yiongem anl bime kornemefre yeams by
   ↪ dathem gake be si"
62 Likelihood (Iter = 3000): -4.15e+03
63 Decrypted Text (Iter = 3100): "un by yionger anl bire komnerafme years by
   ↪ dather gake be si"
64 Likelihood (Iter = 3100): -4.13e+03
65 Decrypted Text (Iter = 3200): "un by yionger anl bire komnerafme years by
   ↪ dather gake be si"
66 Likelihood (Iter = 3200): -4.13e+03
67 Decrypted Text (Iter = 3300): "un by yionger and bire komnerapme years by
   ↪ lather gake be si"
68 Likelihood (Iter = 3300): -4.09e+03
69 Decrypted Text (Iter = 3400): "un by yionger and bore kimnerapme years by
   ↪ lather gake be so"
70 Likelihood (Iter = 3400): -4.08e+03
71 Decrypted Text (Iter = 3500): "un by yionger and bore kimneralme years by
   ↪ pather gake be so"
72 Likelihood (Iter = 3500): -4.07e+03
73 Decrypted Text (Iter = 3600): "un by yionger and bore vilneramle years by
   ↪ pather gave be so"
74 Likelihood (Iter = 3600): -3.97e+03
75 Decrypted Text (Iter = 3700): "un by yionger and bore vilneramle years by
   ↪ pather gave be so"

```

```

76 Likelihood (Iter = 3700): -3.97e+03
77 Decrypted Text (Iter = 3800): "un by yoinger and bore vilneramle years by
   ↳ pather gave be so"
78 Likelihood (Iter = 3800): -3.97e+03
79 Decrypted Text (Iter = 3900): "un by yoinger and bore pilneramle years by
   ↳ vather gape be so"
80 Likelihood (Iter = 3900): -3.97e+03
81 Decrypted Text (Iter = 4000): "un by yoinger and bore pilneramle years by
   ↳ vather gape be so"
82 Likelihood (Iter = 4000): -3.97e+03
83 Decrypted Text (Iter = 4100): "un by yoinger and bore pilneramle years by
   ↳ vather gape be so"
84 Likelihood (Iter = 4100): -3.96e+03
85 Decrypted Text (Iter = 4200): "un by yoinger and bore pilneramle years by
   ↳ vather gape be so"
86 Likelihood (Iter = 4200): -3.96e+03
87 Decrypted Text (Iter = 4300): "un by yoinger and bore pilneramle years by
   ↳ vather gape be so"
88 Likelihood (Iter = 4300): -3.96e+03
89 Decrypted Text (Iter = 4400): "in by younger and bore pulneramle years by
   ↳ vather gape be so"
90 Likelihood (Iter = 4400): -3.79e+03
91 Decrypted Text (Iter = 4500): "in by younger and bore pulneramle years by
   ↳ vather gape be so"
92 Likelihood (Iter = 4500): -3.79e+03
93 Decrypted Text (Iter = 4600): "in by younger and bore vulneramle years by
   ↳ pather gave be so"
94 Likelihood (Iter = 4600): -3.76e+03
95 Decrypted Text (Iter = 4700): "in by younger and bore vulneramle years by
   ↳ pather gave be so"
96 Likelihood (Iter = 4700): -3.76e+03
97 Decrypted Text (Iter = 4800): "in by younger and bore vulneramle years by
   ↳ pather gave be so"
98 Likelihood (Iter = 4800): -3.75e+03
99 Decrypted Text (Iter = 4900): "in by younger and bore vulneramle years by
   ↳ pather gave be so"
100 Likelihood (Iter = 4900): -3.75e+03
101 Decrypted Text (Iter = 5000): "in by younger and bore vulneramle years by
   ↳ pather gave be so"
102 Likelihood (Iter = 5000): -3.75e+03
103 Decrypted Text (Iter = 5100): "in by younger and bore vulneramle years by
   ↳ pather gave be so"
104 Likelihood (Iter = 5100): -3.75e+03
105 Decrypted Text (Iter = 5200): "in my younger and more vulnerable years my
   ↳ pather gave me so"
106 Likelihood (Iter = 5200): -3.68e+03

```

```
107 Decrypted Text (Iter = 5300): "in my younger and more vulnerable years my
    ↳ pather gave me so"
108 Likelihood (Iter = 5300): -3.68e+03
109 Decrypted Text (Iter = 5400): "in my younger and more vulnerable years my
    ↳ pather gave me so"
110 Likelihood (Iter = 5400): -3.68e+03
111 Decrypted Text (Iter = 5500): "in my younger and more vulnerable years my
    ↳ pather gave me so"
112 Likelihood (Iter = 5500): -3.68e+03
113 Decrypted Text (Iter = 5600): "in my younger and more vulnerable years my
    ↳ pather gave me so"
114 Likelihood (Iter = 5600): -3.68e+03
115 Decrypted Text (Iter = 5700): "in my younger and more vulnerable years my
    ↳ pather gave me so"
116 Likelihood (Iter = 5700): -3.68e+03
117 Decrypted Text (Iter = 5800): "in my younger and more vulnerable years my
    ↳ pather gave me so"
118 Likelihood (Iter = 5800): -3.68e+03
119 Decrypted Text (Iter = 5900): "in my younger and more vulnerable years my
    ↳ pather gave me so"
120 Likelihood (Iter = 5900): -3.68e+03
121 Decrypted Text (Iter = 6000): "in my younger and more vulnerable years my
    ↳ pather gave me so"
122 Likelihood (Iter = 6000): -3.68e+03
123 Decrypted Text (Iter = 6100): "in my younger and more vulnerable years my
    ↳ pather gave me so"
124 Likelihood (Iter = 6100): -3.68e+03
125 Decrypted Text (Iter = 6200): "in my younger and more vulnerable years my
    ↳ pather gave me so"
126 Likelihood (Iter = 6200): -3.68e+03
127 Decrypted Text (Iter = 6300): "in my younger and more vulnerable years my
    ↳ pather gave me so"
128 Likelihood (Iter = 6300): -3.68e+03
129 Decrypted Text (Iter = 6400): "in my younger and more vulnerable years my
    ↳ pather gave me so"
130 Likelihood (Iter = 6400): -3.66e+03
131 Decrypted Text (Iter = 6500): "in my younger and more vulnerable years my
    ↳ pather gave me so"
132 Likelihood (Iter = 6500): -3.66e+03
133 Decrypted Text (Iter = 6600): "in my younger and more vulnerable years my
    ↳ pather gave me so"
134 Likelihood (Iter = 6600): -3.66e+03
135 Decrypted Text (Iter = 6700): "in my younger and more vulnerable years my
    ↳ pather gave me so"
136 Likelihood (Iter = 6700): -3.66e+03
137 Decrypted Text (Iter = 6800): "in my younger and more vulnerable years my
    ↳ pather gave me so"
```

```
138 Likelihood (Iter = 6800): -3.66e+03
139 Decrypted Text (Iter = 6900): "in my younger and more vulnerable years my
    ↳ pather gave me so"
140 Likelihood (Iter = 6900): -3.66e+03
141 Decrypted Text (Iter = 7000): "in my younger and more vulnerable years my
    ↳ pather gave me so"
142 Likelihood (Iter = 7000): -3.66e+03
143 Decrypted Text (Iter = 7100): "in my younger and more vulnerable years my
    ↳ pather gave me so"
144 Likelihood (Iter = 7100): -3.66e+03
145 Decrypted Text (Iter = 7200): "in my younger and more vulnerable years my
    ↳ pather gave me so"
146 Likelihood (Iter = 7200): -3.66e+03
147 Decrypted Text (Iter = 7300): "in my younger and more vulnerable years my
    ↳ pather gave me so"
148 Likelihood (Iter = 7300): -3.66e+03
149 Decrypted Text (Iter = 7400): "in my younger and more vulnerable years my
    ↳ pather gave me so"
150 Likelihood (Iter = 7400): -3.66e+03
151 Decrypted Text (Iter = 7500): "in my younger and more vulnerable years my
    ↳ pather gave me so"
152 Likelihood (Iter = 7500): -3.66e+03
153 Decrypted Text (Iter = 7600): "in my younger and more vulnerable years my
    ↳ pather gave me so"
154 Likelihood (Iter = 7600): -3.66e+03
155 Decrypted Text (Iter = 7700): "in my younger and more vulnerable years my
    ↳ pather gave me so"
156 Likelihood (Iter = 7700): -3.66e+03
157 Decrypted Text (Iter = 7800): "in my younger and more vulnerable years my
    ↳ pather gave me so"
158 Likelihood (Iter = 7800): -3.66e+03
159 Decrypted Text (Iter = 7900): "in my younger and more vulnerable years my
    ↳ pather gave me so"
160 Likelihood (Iter = 7900): -3.66e+03
161 Decrypted Text (Iter = 8000): "in my younger and more vulnerable years my
    ↳ father gave me so"
162 Likelihood (Iter = 8000): -3.62e+03
163 Decrypted Text (Iter = 8100): "in my younger and more vulnerable years my
    ↳ father gave me so"
164 Likelihood (Iter = 8100): -3.62e+03
165 Decrypted Text (Iter = 8200): "in my younger and more vulnerable years my
    ↳ father gave me so"
166 Likelihood (Iter = 8200): -3.62e+03
167 Decrypted Text (Iter = 8300): "in my younger and more vulnerable years my
    ↳ father gave me so"
168 Likelihood (Iter = 8300): -3.61e+03
```

```
169 Decrypted Text (Iter = 8400): "in my younger and more vulnerable years my
    ↳ father gave me so"
170 Likelihood (Iter = 8400): -3.61e+03
171 Decrypted Text (Iter = 8500): "in my younger and more vulnerable years my
    ↳ father gave me so"
172 Likelihood (Iter = 8500): -3.61e+03
173 Decrypted Text (Iter = 8600): "in my younger and more vulnerable years my
    ↳ father gave me so"
174 Likelihood (Iter = 8600): -3.61e+03
175 Decrypted Text (Iter = 8700): "in my younger and more vulnerable years my
    ↳ father gave me so"
176 Likelihood (Iter = 8700): -3.61e+03
177 Decrypted Text (Iter = 8800): "in my younger and more vulnerable years my
    ↳ father gave me so"
178 Likelihood (Iter = 8800): -3.61e+03
179 Decrypted Text (Iter = 8900): "in my younger and more vulnerable years my
    ↳ father gave me so"
180 Likelihood (Iter = 8900): -3.61e+03
181 Decrypted Text (Iter = 9000): "in my younger and more vulnerable years my
    ↳ father gave me so"
182 Likelihood (Iter = 9000): -3.61e+03
183 Decrypted Text (Iter = 9100): "in my younger and more vulnerable years my
    ↳ father gave me so"
184 Likelihood (Iter = 9100): -3.61e+03
185 Decrypted Text (Iter = 9200): "in my younger and more vulnerable years my
    ↳ father gave me so"
186 Likelihood (Iter = 9200): -3.61e+03
187 Decrypted Text (Iter = 9300): "in my younger and more vulnerable years my
    ↳ father gave me so"
188 Likelihood (Iter = 9300): -3.61e+03
189 Decrypted Text (Iter = 9400): "in my younger and more vulnerable years my
    ↳ father gave me so"
190 Likelihood (Iter = 9400): -3.61e+03
191 Decrypted Text (Iter = 9500): "in my younger and more vulnerable years my
    ↳ father gave me so"
192 Likelihood (Iter = 9500): -3.61e+03
193 Decrypted Text (Iter = 9600): "in my younger and more vulnerable years my
    ↳ father gave me so"
194 Likelihood (Iter = 9600): -3.61e+03
195 Decrypted Text (Iter = 9700): "in my younger and more vulnerable years my
    ↳ father gave me so"
196 Likelihood (Iter = 9700): -3.61e+03
197 Decrypted Text (Iter = 9800): "in my younger and more vulnerable years my
    ↳ father gave me so"
198 Likelihood (Iter = 9800): -3.61e+03
199 Decrypted Text (Iter = 9900): "in my younger and more vulnerable years my
    ↳ father gave me so"
```

200 Likelihood (Iter = 9900): -3.61e+03

201 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...

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

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...

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

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

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

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 $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$.

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 $\text{span}\{\xi_1, \dots, \xi_k\}$, implies $R_A(x) < \lambda_1$

First, recall that since $\{\xi_1, \dots, \xi_n\}$ 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 $\text{span}\{\xi_1, \dots, \xi_k\}$, 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 $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$ and x has non-zero components in $\text{span}\{\xi_{k+1}, \dots, \xi_n\}$, 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 $\text{span}\{\xi_1, \dots, \xi_k\}$).

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

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...

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

Result:

```
1 Symbol: =
2 Marginal Probability p('='): 6.34680665183418e-07
3 Transition Probabilities:
4 p('=|'=): 0.0
5 p(' '|'=): 1.0
6 p('-|'=): 0.0
7 p(',|'=): 0.0
8 p(';|'=): 0.0
9 p(':',|'=): 0.0
10 p('!|'=): 0.0
11 p('?|'=): 0.0
12 p('/|'=): 0.0
13 p('.|'=): 0.0
14 p(''|'=): 0.0
15 p('"'|'=): 0.0
16 p('('|'=): 0.0
17 p(')'|'=): 0.0
18 p('[|'=): 0.0
19 p(']|'=): 0.0
20 p('*|'=): 0.0
21 p('0|'=): 0.0
22 p('1|'=): 0.0
23 p('2|'=): 0.0
24 p('3|'=): 0.0
```

```

25 p('4'|'='): 0.0
26 p('5'|'='): 0.0
27 p('6'|'='): 0.0
28 p('7'|'='): 0.0
29 p('8'|'='): 0.0
30 p('9'|'='): 0.0
31 p('a'|'='): 0.0
32 p('b'|'='): 0.0
33 p('c'|'='): 0.0
34 p('d'|'='): 0.0
35 p('e'|'='): 0.0
36 p('f'|'='): 0.0
37 p('g'|'='): 0.0
38 p('h'|'='): 0.0
39 p('i'|'='): 0.0
40 p('j'|'='): 0.0
41 p('k'|'='): 0.0
42 p('l'|'='): 0.0
43 p('m'|'='): 0.0
44 p('n'|'='): 0.0
45 p('o'|'='): 0.0
46 p('p'|'='): 0.0
47 p('q'|'='): 0.0
48 p('r'|'='): 0.0
49 p('s'|'='): 0.0
50 p('t'|'='): 0.0
51 p('u'|'='): 0.0
52 p('v'|'='): 0.0
53 p('w'|'='): 0.0
54 p('x'|'='): 0.0
55 p('y'|'='): 0.0
56 p('z'|'='): 0.0
57
58
59 Symbol:
60 Marginal Probability p(' '): 0.18207718922781893
61 Transition Probabilities:
62 p('='|' '): 3.485784100990137e-06
63 p(' '|' '): 0.029043553129449822
64 p('-'|' '): 2.7886272807921097e-05
65 p(', '|' '): 0.0
66 p('; '|' '): 0.0
67 p(':', '|' '): 0.0
68 p('!'|' '): 0.0
69 p('? '|' '): 0.0
70 p('/ '|' '): 0.0
71 p('. '|' '): 3.485784100990137e-05

```

```

72 p(''|' '): 1.7428920504950686e-06
73 p('"'|' '): 1.7428920504950685e-05
74 p('('|' '): 0.0011259082646198142
75 p(')|' '): 0.0
76 p('['|' '): 3.485784100990137e-06
77 p(']|' '): 0.0
78 p('*'|' '): 0.0004792953138861438
79 p('0'|' '): 3.485784100990137e-06
80 p('1'|' '): 0.0004426945808257474
81 p('2'|' '): 4.531519331287178e-05
82 p('3'|' '): 2.7886272807921097e-05
83 p('4'|' '): 1.5686028454455618e-05
84 p('5'|' '): 1.220024435346548e-05
85 p('6'|' '): 3.1372056908911235e-05
86 p('7'|' '): 1.220024435346548e-05
87 p('8'|' '): 1.3943136403960548e-05
88 p('9'|' '): 1.220024435346548e-05
89 p('a'|' '): 0.11948570741373991
90 p('b'|' '): 0.04295008880034997
91 p('c'|' '): 0.036741907316486536
92 p('d'|' '): 0.02952807711948745
93 p('e'|' '): 0.02024020538239923
94 p('f'|' '): 0.036935368334091494
95 p('g'|' '): 0.016728277900651668
96 p('h'|' '): 0.08569625923079202
97 p('i'|' '): 0.05140834392140254
98 p('j'|' '): 0.002818256445650526
99 p('k'|' '): 0.006940196145071363
100 p('l'|' '): 0.02269594028154678
101 p('m'|' '): 0.03355764354023205
102 p('n'|' '): 0.02548456756233889
103 p('o'|' '): 0.05860474519789668
104 p('p'|' '): 0.03264611099782313
105 p('q'|' '): 0.0024557348991475513
106 p('r'|' '): 0.026263640308910187
107 p('s'|' '): 0.07040761016384928
108 p('t'|' '): 0.15062421678788482
109 p('u'|' '): 0.009671307988197136
110 p('v'|' '): 0.006814707917435717
111 p('w'|' '): 0.06848345734010272
112 p('x'|' '): 0.0007825585306722858
113 p('y'|' '): 0.01043295181426348
114 p('z'|' '): 0.00024749067117029973
115
116
117 Symbol: -
118 Marginal Probability p('-'): 0.000580098127977644

```

```

119 Transition Probabilities:
120 p('='|'-'): 0.0
121 p(' '|'-'): 0.0087527352297593
122 p('-'|'-'): 0.002735229759299781
123 p(', '|'-'): 0.0
124 p('; '|'-'): 0.0
125 p(':', '|'-'): 0.0
126 p('! '|'-'): 0.0
127 p('? '|'-'): 0.0
128 p('/ '|'-'): 0.0
129 p('. '|'-'): 0.0
130 p('' '|'-'): 0.0
131 p('"' '|'-'): 0.0
132 p('(' '|'-'): 0.0
133 p(')' '|'-'): 0.0
134 p('[' '|'-'): 0.0
135 p(']' '|'-'): 0.0
136 p('* '|'-'): 0.0
137 p('0 '|'-'): 0.0
138 p('1 '|'-'): 0.0005470459518599562
139 p('2 '|'-'): 0.0
140 p('3 '|'-'): 0.0
141 p('4 '|'-'): 0.0
142 p('5 '|'-'): 0.0
143 p('6 '|'-'): 0.0010940919037199124
144 p('7 '|'-'): 0.0
145 p('8 '|'-'): 0.0005470459518599562
146 p('9 '|'-'): 0.0
147 p('a '|'-'): 0.026805251641137857
148 p('b '|'-'): 0.06345733041575492
149 p('c '|'-'): 0.09573304157549234
150 p('d '|'-'): 0.08315098468271334
151 p('e '|'-'): 0.04266958424507659
152 p('f '|'-'): 0.07056892778993436
153 p('g '|'-'): 0.019146608315098467
154 p('h '|'-'): 0.03938730853391685
155 p('i '|'-'): 0.048687089715536105
156 p('j '|'-'): 0.0005470459518599562
157 p('k '|'-'): 0.019693654266958426
158 p('l '|'-'): 0.06455142231947483
159 p('m '|'-'): 0.03665207877461707
160 p('n '|'-'): 0.03665207877461707
161 p('o '|'-'): 0.05087527352297593
162 p('p '|'-'): 0.028446389496717725
163 p('q '|'-'): 0.0005470459518599562
164 p('r '|'-'): 0.02899343544857768
165 p('s '|'-'): 0.09026258205689278

```

```

166 p('t'|'-'): 0.07221006564551423
167 p('u'|'-'): 0.016411378555798686
168 p('v'|'-'): 0.0032822757111597373
169 p('w'|'-'): 0.023522975929978117
170 p('x'|'-'): 0.0
171 p('y'|'-'): 0.0175054704595186
172 p('z'|'-'): 0.006564551422319475
173
174
175 Symbol: ,
176 Marginal Probability p(','): 0.011806647074074533
177 Transition Probabilities:
178 p('= '|','): 0.0
179 p(' '|','): 0.9996774627066255
180 p('- '|','): 0.0
181 p(', '|','): 0.0
182 p('; '|','): 0.0
183 p(': '|','): 0.0
184 p('! '|','): 0.0
185 p('? '|','): 0.0
186 p('/ '|','): 0.0
187 p('. '|','): 2.6878107781212204e-05
188 p('' '|','): 0.0
189 p("'" '|','): 2.6878107781212204e-05
190 p('( '|','): 0.0
191 p(')' '|','): 0.0
192 p('[ '|','): 0.0
193 p('] '|','): 0.0
194 p('* '|','): 2.6878107781212204e-05
195 p('0 '|','): 0.00024190297003090982
196 p('1 '|','): 0.0
197 p('2 '|','): 0.0
198 p('3 '|','): 0.0
199 p('4 '|','): 0.0
200 p('5 '|','): 0.0
201 p('6 '|','): 0.0
202 p('7 '|','): 0.0
203 p('8 '|','): 0.0
204 p('9 '|','): 0.0
205 p('a '|','): 0.0
206 p('b '|','): 0.0
207 p('c '|','): 0.0
208 p('d '|','): 0.0
209 p('e '|','): 0.0
210 p('f '|','): 0.0
211 p('g '|','): 0.0
212 p('h '|','): 0.0

```

```

213 p('i'|','): 0.0
214 p('j'|','): 0.0
215 p('k'|','): 0.0
216 p('l'|','): 0.0
217 p('m'|','): 0.0
218 p('n'|','): 0.0
219 p('o'|','): 0.0
220 p('p'|','): 0.0
221 p('q'|','): 0.0
222 p('r'|','): 0.0
223 p('s'|','): 0.0
224 p('t'|','): 0.0
225 p('u'|','): 0.0
226 p('v'|','): 0.0
227 p('w'|','): 0.0
228 p('x'|','): 0.0
229 p('y'|','): 0.0
230 p('z'|','): 0.0
231
232
233 Symbol: ;
234 Marginal Probability p(';'): 0.0003630373404849151
235 Transition Probabilities:
236 p('= '|';'): 0.0
237 p(' '|';'): 1.0
238 p('- '|';'): 0.0
239 p(', '|';'): 0.0
240 p('; '|';'): 0.0
241 p(': '|';'): 0.0
242 p('! '|';'): 0.0
243 p('? '|';'): 0.0
244 p('/ '|';'): 0.0
245 p('. '|';'): 0.0
246 p('' '|';'): 0.0
247 p('" '|';'): 0.0
248 p('(' '|';'): 0.0
249 p(')' '|';'): 0.0
250 p('[' '|';'): 0.0
251 p(']' '|';'): 0.0
252 p('* '|';'): 0.0
253 p('0 '|';'): 0.0
254 p('1 '|';'): 0.0
255 p('2 '|';'): 0.0
256 p('3 '|';'): 0.0
257 p('4 '|';'): 0.0
258 p('5 '|';'): 0.0
259 p('6 '|';'): 0.0

```

```

260 p('7'|';'): 0.0
261 p('8'|';'): 0.0
262 p('9'|';'): 0.0
263 p('a'|';'): 0.0
264 p('b'|';'): 0.0
265 p('c'|';'): 0.0
266 p('d'|';'): 0.0
267 p('e'|';'): 0.0
268 p('f'|';'): 0.0
269 p('g'|';'): 0.0
270 p('h'|';'): 0.0
271 p('i'|';'): 0.0
272 p('j'|';'): 0.0
273 p('k'|';'): 0.0
274 p('l'|';'): 0.0
275 p('m'|';'): 0.0
276 p('n'|';'): 0.0
277 p('o'|';'): 0.0
278 p('p'|';'): 0.0
279 p('q'|';'): 0.0
280 p('r'|';'): 0.0
281 p('s'|';'): 0.0
282 p('t'|';'): 0.0
283 p('u'|';'): 0.0
284 p('v'|';'): 0.0
285 p('w'|';'): 0.0
286 p('x'|';'): 0.0
287 p('y'|';'): 0.0
288 p('z'|';'): 0.0
289
290
291 Symbol: :
292 Marginal Probability p(':'): 0.00031956171491985097
293 Transition Probabilities:
294 p('='|':'): 0.0
295 p(' '|':'): 0.9990069513406157
296 p('-'|':'): 0.0
297 p(', '|':'): 0.0
298 p('; '|':'): 0.0
299 p(': '|':'): 0.0
300 p('! '|':'): 0.0
301 p('? '|':'): 0.0
302 p('/ '|':'): 0.0
303 p('. '|':'): 0.0
304 p('' '|':'): 0.0
305 p('" '|':'): 0.0
306 p('(' '|':'): 0.0

```

```

307 p(')|':'): 0.0
308 p('['|':'): 0.0
309 p(']|':'): 0.0
310 p('*'|':'): 0.0009930486593843098
311 p('0'|':'): 0.0
312 p('1'|':'): 0.0
313 p('2'|':'): 0.0
314 p('3'|':'): 0.0
315 p('4'|':'): 0.0
316 p('5'|':'): 0.0
317 p('6'|':'): 0.0
318 p('7'|':'): 0.0
319 p('8'|':'): 0.0
320 p('9'|':'): 0.0
321 p('a'|':'): 0.0
322 p('b'|':'): 0.0
323 p('c'|':'): 0.0
324 p('d'|':'): 0.0
325 p('e'|':'): 0.0
326 p('f'|':'): 0.0
327 p('g'|':'): 0.0
328 p('h'|':'): 0.0
329 p('i'|':'): 0.0
330 p('j'|':'): 0.0
331 p('k'|':'): 0.0
332 p('l'|':'): 0.0
333 p('m'|':'): 0.0
334 p('n'|':'): 0.0
335 p('o'|':'): 0.0
336 p('p'|':'): 0.0
337 p('q'|':'): 0.0
338 p('r'|':'): 0.0
339 p('s'|':'): 0.0
340 p('t'|':'): 0.0
341 p('u'|':'): 0.0
342 p('v'|':'): 0.0
343 p('w'|':'): 0.0
344 p('x'|':'): 0.0
345 p('y'|':'): 0.0
346 p('z'|':'): 0.0
347
348
349 Symbol: !
350 Marginal Probability p('!'): 0.0005791461069798689
351 Transition Probabilities:
352 p('=|!'): 0.0
353 p(' '|!'): 0.9019178082191781

```



```

354 p('-'|'!'): 0.0
355 p(', '|'!'): 0.0
356 p('; '|'!'): 0.0
357 p(': '|'!'): 0.0
358 p('! '|'!'): 0.0
359 p('? '|'!'): 0.0
360 p('/ '|'!'): 0.0
361 p('. '|'!'): 0.0958904109589041
362 p(''|'!'): 0.0
363 p('"'|'!'): 0.0
364 p('('|'!'): 0.0
365 p(')'|'!'): 0.0016438356164383563
366 p('['|'!'): 0.0
367 p(']'|'!'): 0.0
368 p('* '|'!'): 0.000547945205479452
369 p('0 '|'!'): 0.0
370 p('1 '|'!'): 0.0
371 p('2 '|'!'): 0.0
372 p('3 '|'!'): 0.0
373 p('4 '|'!'): 0.0
374 p('5 '|'!'): 0.0
375 p('6 '|'!'): 0.0
376 p('7 '|'!'): 0.0
377 p('8 '|'!'): 0.0
378 p('9 '|'!'): 0.0
379 p('a '|'!'): 0.0
380 p('b '|'!'): 0.0
381 p('c '|'!'): 0.0
382 p('d '|'!'): 0.0
383 p('e '|'!'): 0.0
384 p('f '|'!'): 0.0
385 p('g '|'!'): 0.0
386 p('h '|'!'): 0.0
387 p('i '|'!'): 0.0
388 p('j '|'!'): 0.0
389 p('k '|'!'): 0.0
390 p('l '|'!'): 0.0
391 p('m '|'!'): 0.0
392 p('n '|'!'): 0.0
393 p('o '|'!'): 0.0
394 p('p '|'!'): 0.0
395 p('q '|'!'): 0.0
396 p('r '|'!'): 0.0
397 p('s '|'!'): 0.0
398 p('t '|'!'): 0.0
399 p('u '|'!'): 0.0
400 p('v '|'!'): 0.0

```

```

401 p('w'|'!'): 0.0
402 p('x'|'!'): 0.0
403 p('y'|'!'): 0.0
404 p('z'|'!'): 0.0
405
406
407 Symbol: ?
408 Marginal Probability p('?'): 0.0004515752932780019
409 Transition Probabilities:
410 p('='|'?'): 0.0
411 p(' '|'?'): 0.9163738580463809
412 p('-'|'?'): 0.0
413 p(', '|'?'): 0.0
414 p('; '|'?'): 0.0
415 p(':', '|'?'): 0.0
416 p('!'|'?'): 0.0
417 p('? '|'?'): 0.0
418 p('/ '|'?'): 0.0
419 p('. '|'?'): 0.08222066057624737
420 p('' '|'?'): 0.0
421 p('"'|'?'): 0.0
422 p('('|'?'): 0.0
423 p(')'|'?'): 0.0014054813773717498
424 p('['|'?'): 0.0
425 p(']'|'?'): 0.0
426 p('*'|'?'): 0.0
427 p('0'|'?'): 0.0
428 p('1'|'?'): 0.0
429 p('2'|'?'): 0.0
430 p('3'|'?'): 0.0
431 p('4'|'?'): 0.0
432 p('5'|'?'): 0.0
433 p('6'|'?'): 0.0
434 p('7'|'?'): 0.0
435 p('8'|'?'): 0.0
436 p('9'|'?'): 0.0
437 p('a'|'?'): 0.0
438 p('b'|'?'): 0.0
439 p('c'|'?'): 0.0
440 p('d'|'?'): 0.0
441 p('e'|'?'): 0.0
442 p('f'|'?'): 0.0
443 p('g'|'?'): 0.0
444 p('h'|'?'): 0.0
445 p('i'|'?'): 0.0
446 p('j'|'?'): 0.0
447 p('k'|'?'): 0.0

```

```

448 p('l'|'?'): 0.0
449 p('m'|'?'): 0.0
450 p('n'|'?'): 0.0
451 p('o'|'?'): 0.0
452 p('p'|'?'): 0.0
453 p('q'|'?'): 0.0
454 p('r'|'?'): 0.0
455 p('s'|'?'): 0.0
456 p('t'|'?'): 0.0
457 p('u'|'?'): 0.0
458 p('v'|'?'): 0.0
459 p('w'|'?'): 0.0
460 p('x'|'?'): 0.0
461 p('y'|'?'): 0.0
462 p('z'|'?'): 0.0
463
464
465 Symbol: /
466 Marginal Probability p('/'): 2.856062993325381e-06
467 Transition Probabilities:
468 p('=|'/'): 0.0
469 p(' '|'/'): 0.0
470 p('-|'/'): 0.0
471 p(',|'/'): 0.0
472 p(';|'/'): 0.0
473 p(':|'/'): 0.0
474 p('!|'/'): 0.0
475 p('?|'/'): 0.0
476 p('/|'/'): 0.0
477 p('.|'/'): 0.0
478 p(''|'/'): 0.0
479 p('"'|'/'): 0.0
480 p('( |'/'): 0.0
481 p(')|'/'): 0.0
482 p('[|'/'): 0.0
483 p(']|'/'): 0.0
484 p('*|'/'): 0.0
485 p('0|'/'): 0.0
486 p('1|'/'): 0.0
487 p('2|'/'): 0.0
488 p('3|'/'): 0.0
489 p('4|'/'): 0.1111111111111111
490 p('5|'/'): 0.0
491 p('6|'/'): 0.0
492 p('7|'/'): 0.0
493 p('8|'/'): 0.0
494 p('9|'/'): 0.0

```

```

495 p('a'|'/'): 0.0
496 p('b'|'/'): 0.0
497 p('c'|'/'): 0.2222222222222222
498 p('d'|'/'): 0.2222222222222222
499 p('e'|'/'): 0.0
500 p('f'|'/'): 0.0
501 p('g'|'/'): 0.0
502 p('h'|'/'): 0.1111111111111111
503 p('i'|'/'): 0.0
504 p('j'|'/'): 0.0
505 p('k'|'/'): 0.0
506 p('l'|'/'): 0.1111111111111111
507 p('m'|'/'): 0.0
508 p('n'|'/'): 0.0
509 p('o'|'/'): 0.0
510 p('p'|'/'): 0.0
511 p('q'|'/'): 0.0
512 p('r'|'/'): 0.0
513 p('s'|'/'): 0.0
514 p('t'|'/'): 0.1111111111111111
515 p('u'|'/'): 0.0
516 p('v'|'/'): 0.0
517 p('w'|'/'): 0.0
518 p('x'|'/'): 0.0
519 p('y'|'/'): 0.1111111111111111
520 p('z'|'/'): 0.0
521
522
523 Symbol: .
524 Marginal Probability p('.'): 0.009049911604850356
525 Transition Probabilities:
526 p('=|'.'): 0.0
527 p(' '|'.'): 0.8526193982747738
528 p('-|'.'): 0.0
529 p(',|'.'): 0.0002805245809664072
530 p(';|'.'): 0.0
531 p(':',|'.'): 0.0
532 p('!|'.'): 0.0003857212988288099
533 p('?|'.'): 0.0005610491619328144
534 p('/|'.'): 0.0
535 p('.|'.'): 0.13994670032961637
536 p(''|'.'): 0.0
537 p('"'|'.'): 3.50655726208009e-05
538 p('(|'.'): 0.0
539 p(')|'.'): 0.002454590083456063
540 p('[|'.'): 0.0
541 p(']|'.'): 0.0

```

```

542 p('*'|'.'): 3.50655726208009e-05
543 p('0'|'.'): 0.0
544 p('1'|'.'): 0.00021039343572480537
545 p('2'|'.'): 7.01311452416018e-05
546 p('3'|'.'): 0.00021039343572480537
547 p('4'|'.'): 7.01311452416018e-05
548 p('5'|'.'): 7.01311452416018e-05
549 p('6'|'.'): 7.01311452416018e-05
550 p('7'|'.'): 0.00010519671786240269
551 p('8'|'.'): 0.0001402622904832036
552 p('9'|'.'): 0.00010519671786240269
553 p('a'|'.'): 0.0001402622904832036
554 p('b'|'.'): 3.50655726208009e-05
555 p('c'|'.'): 7.01311452416018e-05
556 p('d'|'.'): 3.50655726208009e-05
557 p('e'|'.'): 0.0007714425976576198
558 p('f'|'.'): 0.0003857212988288099
559 p('g'|'.'): 0.0003155901535872081
560 p('h'|'.'): 0.0
561 p('i'|'.'): 0.0
562 p('j'|'.'): 0.0
563 p('k'|'.'): 0.0
564 p('l'|'.'): 3.50655726208009e-05
565 p('m'|'.'): 0.0
566 p('n'|'.'): 0.00010519671786240269
567 p('o'|'.'): 0.000350655726208009
568 p('p'|'.'): 0.0
569 p('q'|'.'): 0.0
570 p('r'|'.'): 0.0
571 p('s'|'.'): 0.0003155901535872081
572 p('t'|'.'): 3.50655726208009e-05
573 p('u'|'.'): 3.50655726208009e-05
574 p('v'|'.'): 0.0
575 p('w'|'.'): 0.0
576 p('x'|'.'): 0.0
577 p('y'|'.'): 0.0
578 p('z'|'.'): 0.0
579
580
581 Symbol: '
582 Marginal Probability p(''): 2.221382328141963e-06
583 Transition Probabilities:
584 p('=|'''): 0.0
585 p(' '|'''): 0.0
586 p('-|'''): 0.0
587 p(',|'''): 0.14285714285714285
588 p(';|'''): 0.0

```

```

589 p(':', '|'): 0.0
590 p('!' | '|'): 0.0
591 p('? | '|'): 0.0
592 p('/ | '|'): 0.0
593 p('. | '|'): 0.0
594 p('' | '|'): 0.0
595 p('" | '|'): 0.0
596 p('(' | '|'): 0.0
597 p(')' | '|'): 0.0
598 p('[' | '|'): 0.0
599 p(']' | '|'): 0.0
600 p('* | '|'): 0.0
601 p('0' | '|'): 0.0
602 p('1' | '|'): 0.0
603 p('2' | '|'): 0.0
604 p('3' | '|'): 0.0
605 p('4' | '|'): 0.0
606 p('5' | '|'): 0.0
607 p('6' | '|'): 0.0
608 p('7' | '|'): 0.0
609 p('8' | '|'): 0.0
610 p('9' | '|'): 0.0
611 p('a' | '|'): 0.14285714285714285
612 p('b' | '|'): 0.0
613 p('c' | '|'): 0.0
614 p('d' | '|'): 0.0
615 p('e' | '|'): 0.0
616 p('f' | '|'): 0.0
617 p('g' | '|'): 0.0
618 p('h' | '|'): 0.0
619 p('i' | '|'): 0.0
620 p('j' | '|'): 0.0
621 p('k' | '|'): 0.0
622 p('l' | '|'): 0.0
623 p('m' | '|'): 0.0
624 p('n' | '|'): 0.0
625 p('o' | '|'): 0.0
626 p('p' | '|'): 0.0
627 p('q' | '|'): 0.0
628 p('r' | '|'): 0.0
629 p('s' | '|'): 0.7142857142857143
630 p('t' | '|'): 0.0
631 p('u' | '|'): 0.0
632 p('v' | '|'): 0.0
633 p('w' | '|'): 0.0
634 p('x' | '|'): 0.0
635 p('y' | '|'): 0.0

```

```

636 p('z'|''): 0.0
637
638
639 Symbol: "
640 Marginal Probability p(''): 6.9814873170175976e-06
641 Transition Probabilities:
642 p('=|'''): 0.0
643 p(' '|'''): 0.45454545454545453
644 p('-|'''): 0.0
645 p(',|'''): 0.0
646 p(';|'''): 0.0
647 p(':',|'''): 0.0
648 p('!|'''): 0.0
649 p('?|'''): 0.0
650 p('/|'''): 0.0
651 p('.|'''): 0.0
652 p(''|'''): 0.0
653 p('"'|'''): 0.0
654 p('(|'''): 0.0
655 p(')|'''): 0.045454545454545456
656 p('[|'''): 0.0
657 p(']|'''): 0.0
658 p('*|'''): 0.0
659 p('0|'''): 0.0
660 p('1|'''): 0.0
661 p('2|'''): 0.0
662 p('3|'''): 0.0
663 p('4|'''): 0.0
664 p('5|'''): 0.0
665 p('6|'''): 0.0
666 p('7|'''): 0.0
667 p('8|'''): 0.0
668 p('9|'''): 0.0
669 p('a|'''): 0.0
670 p('b|'''): 0.0
671 p('c|'''): 0.0
672 p('d|'''): 0.045454545454545456
673 p('e|'''): 0.0
674 p('f|'''): 0.0
675 p('g|'''): 0.0
676 p('h|'''): 0.0
677 p('i|'''): 0.045454545454545456
678 p('j|'''): 0.0
679 p('k|'''): 0.0
680 p('l|'''): 0.0
681 p('m|'''): 0.0
682 p('n|'''): 0.0

```

```

683 p('o'|''): 0.0
684 p('p'|''): 0.3181818181818182
685 p('q'|''): 0.0
686 p('r'|''): 0.045454545454545456
687 p('s'|''): 0.0
688 p('t'|''): 0.045454545454545456
689 p('u'|''): 0.0
690 p('v'|''): 0.0
691 p('w'|''): 0.0
692 p('x'|''): 0.0
693 p('y'|''): 0.0
694 p('z'|''): 0.0
695
696
697 Symbol: (
698 Marginal Probability p('('): 0.00020944461951052793
699 Transition Probabilities:
700 p('=|'('): 0.0
701 p(' '|'('): 0.0
702 p('-|'('): 0.0
703 p(',|'('): 0.0
704 p(';|'('): 0.0
705 p(':',|'('): 0.0
706 p('!|'('): 0.0
707 p('?|'('): 0.0
708 p('/|'('): 0.0
709 p('.|'('): 0.0
710 p(''|'('): 0.0
711 p('"'|'('): 0.0015151515151515152
712 p('('|'('): 0.0
713 p(')'|'('): 0.0
714 p('[|'('): 0.0
715 p(']|'('): 0.0
716 p('*|'('): 0.0
717 p('0'|'('): 0.0
718 p('1'|'('): 0.025757575757575757
719 p('2'|'('): 0.05909090909090909
720 p('3'|'('): 0.022727272727272728
721 p('4'|'('): 0.004545454545454545
722 p('5'|'('): 0.0015151515151515152
723 p('6'|'('): 0.0
724 p('7'|'('): 0.0
725 p('8'|'('): 0.0015151515151515152
726 p('9'|'('): 0.0
727 p('a'|'('): 0.12424242424242424
728 p('b'|'('): 0.024242424242424242
729 p('c'|'('): 0.007575757575757576

```



```

730 p('d'|'('): 0.010606060606060607
731 p('e'|'('): 0.010606060606060607
732 p('f'|'('): 0.021212121212121213
733 p('g'|'('): 0.006060606060606061
734 p('h'|'('): 0.11515151515151516
735 p('i'|'('): 0.06515151515151515
736 p('j'|'('): 0.0015151515151515152
737 p('k'|'('): 0.007575757575757576
738 p('l'|'('): 0.00909090909090909
739 p('m'|'('): 0.01818181818181818
740 p('n'|'('): 0.025757575757575757
741 p('o'|'('): 0.05
742 p('p'|'('): 0.03484848484848485
743 p('q'|'('): 0.0015151515151515152
744 p('r'|'('): 0.006060606060606061
745 p('s'|'('): 0.06666666666666667
746 p('t'|'('): 0.15151515151515152
747 p('u'|'('): 0.0030303030303030303
748 p('v'|'('): 0.004545454545454545
749 p('w'|'('): 0.11666666666666667
750 p('x'|'('): 0.0
751 p('y'|'('): 0.0015151515151515152
752 p('z'|'('): 0.0
753
754
755 Symbol: )
756 Marginal Probability p(')'): 0.0002078579178475694
757 Transition Probabilities:
758 p('=|')'): 0.0
759 p(' '|')'): 0.6748091603053435
760 p('-|')'): 0.0
761 p(',|')'): 0.2595419847328244
762 p(';|')'): 0.007633587786259542
763 p(':',|')'): 0.0015267175572519084
764 p('!|')'): 0.0
765 p('?|')'): 0.0
766 p('/|')'): 0.0
767 p('.|')'): 0.0549618320610687
768 p(''|')'): 0.0
769 p('"'|')'): 0.0
770 p('( |')'): 0.0015267175572519084
771 p(')|')'): 0.0
772 p('[|')'): 0.0
773 p(']|')'): 0.0
774 p('*|')'): 0.0
775 p('0|')'): 0.0
776 p('1|')'): 0.0

```

```

777 p('2'|'')): 0.0
778 p('3'|'')): 0.0
779 p('4'|'')): 0.0
780 p('5'|'')): 0.0
781 p('6'|'')): 0.0
782 p('7'|'')): 0.0
783 p('8'|'')): 0.0
784 p('9'|'')): 0.0
785 p('a'|'')): 0.0
786 p('b'|'')): 0.0
787 p('c'|'')): 0.0
788 p('d'|'')): 0.0
789 p('e'|'')): 0.0
790 p('f'|'')): 0.0
791 p('g'|'')): 0.0
792 p('h'|'')): 0.0
793 p('i'|'')): 0.0
794 p('j'|'')): 0.0
795 p('k'|'')): 0.0
796 p('l'|'')): 0.0
797 p('m'|'')): 0.0
798 p('n'|'')): 0.0
799 p('o'|'')): 0.0
800 p('p'|'')): 0.0
801 p('q'|'')): 0.0
802 p('r'|'')): 0.0
803 p('s'|'')): 0.0
804 p('t'|'')): 0.0
805 p('u'|'')): 0.0
806 p('v'|'')): 0.0
807 p('w'|'')): 0.0
808 p('x'|'')): 0.0
809 p('y'|'')): 0.0
810 p('z'|'')): 0.0
811
812
813 Symbol: [
814 Marginal Probability p('['): 6.34680665183418e-07
815 Transition Probabilities:
816 p('='|'['): 0.0
817 p(' '|'['): 0.0
818 p('-'|'['): 0.0
819 p(', '|'['): 0.0
820 p('; '|'['): 0.0
821 p(':', '|'['): 0.0
822 p('!'|'['): 0.0
823 p('? '|'['): 0.0

```

```

824 p('/'|'|'): 0.0
825 p('.'|'|'): 0.0
826 p(''|'|'): 0.0
827 p('"'|'|'): 0.0
828 p('('|'|'): 0.0
829 p(')'|'|'): 0.0
830 p('['|'|'): 0.0
831 p(']'|'|'): 0.0
832 p('*'|'|'): 0.0
833 p('0'|'|'): 0.0
834 p('1'|'|'): 0.0
835 p('2'|'|'): 0.0
836 p('3'|'|'): 0.0
837 p('4'|'|'): 0.0
838 p('5'|'|'): 0.0
839 p('6'|'|'): 0.0
840 p('7'|'|'): 0.0
841 p('8'|'|'): 0.0
842 p('9'|'|'): 0.0
843 p('a'|'|'): 0.0
844 p('b'|'|'): 0.0
845 p('c'|'|'): 0.0
846 p('d'|'|'): 0.0
847 p('e'|'|'): 0.5
848 p('f'|'|'): 0.0
849 p('g'|'|'): 0.0
850 p('h'|'|'): 0.0
851 p('i'|'|'): 0.0
852 p('j'|'|'): 0.0
853 p('k'|'|'): 0.0
854 p('l'|'|'): 0.0
855 p('m'|'|'): 0.5
856 p('n'|'|'): 0.0
857 p('o'|'|'): 0.0
858 p('p'|'|'): 0.0
859 p('q'|'|'): 0.0
860 p('r'|'|'): 0.0
861 p('s'|'|'): 0.0
862 p('t'|'|'): 0.0
863 p('u'|'|'): 0.0
864 p('v'|'|'): 0.0
865 p('w'|'|'): 0.0
866 p('x'|'|'): 0.0
867 p('y'|'|'): 0.0
868 p('z'|'|'): 0.0
869
870

```

```

871 Symbol: ]
872 Marginal Probability p(']'): 6.34680665183418e-07
873 Transition Probabilities:
874 p('=|']'): 0.0
875 p(' '|']'): 1.0
876 p('-|']'): 0.0
877 p(',|']'): 0.0
878 p(';|']'): 0.0
879 p(':',|']'): 0.0
880 p('!|']'): 0.0
881 p('?|']'): 0.0
882 p('/|']'): 0.0
883 p('.|']'): 0.0
884 p(''|']'): 0.0
885 p('"|']'): 0.0
886 p('(|']'): 0.0
887 p(')|']'): 0.0
888 p('[|']'): 0.0
889 p(']|']'): 0.0
890 p('*|']'): 0.0
891 p('0|']'): 0.0
892 p('1|']'): 0.0
893 p('2|']'): 0.0
894 p('3|']'): 0.0
895 p('4|']'): 0.0
896 p('5|']'): 0.0
897 p('6|']'): 0.0
898 p('7|']'): 0.0
899 p('8|']'): 0.0
900 p('9|']'): 0.0
901 p('a|']'): 0.0
902 p('b|']'): 0.0
903 p('c|']'): 0.0
904 p('d|']'): 0.0
905 p('e|']'): 0.0
906 p('f|']'): 0.0
907 p('g|']'): 0.0
908 p('h|']'): 0.0
909 p('i|']'): 0.0
910 p('j|']'): 0.0
911 p('k|']'): 0.0
912 p('l|']'): 0.0
913 p('m|']'): 0.0
914 p('n|']'): 0.0
915 p('o|']'): 0.0
916 p('p|']'): 0.0
917 p('q|']'): 0.0

```

```

918 p('r'|']'): 0.0
919 p('s'|']'): 0.0
920 p('t'|']'): 0.0
921 p('u'|']'): 0.0
922 p('v'|']'): 0.0
923 p('w'|']'): 0.0
924 p('x'|']'): 0.0
925 p('y'|']'): 0.0
926 p('z'|']'): 0.0
927
928
929 Symbol: *
930 Marginal Probability p('*'): 9.075933512122877e-05
931 Transition Probabilities:
932 p('=|'*'): 0.0
933 p(' '|'*'): 0.9020979020979021
934 p('-|'*'): 0.0
935 p(',|'*'): 0.0
936 p(';|'*'): 0.0
937 p(':',|'*'): 0.0
938 p('!|'*'): 0.0
939 p('?|'*'): 0.0
940 p('/|'*'): 0.0
941 p('.|'*'): 0.0
942 p(''|'*'): 0.0
943 p('"'|'*'): 0.0
944 p('(|'*'): 0.055944055944055944
945 p(')|'*'): 0.0
946 p('[|'*'): 0.0
947 p(']|'*'): 0.0
948 p('*|'*'): 0.027972027972027972
949 p('0|'*'): 0.0
950 p('1|'*'): 0.0
951 p('2|'*'): 0.0
952 p('3|'*'): 0.0
953 p('4|'*'): 0.0
954 p('5|'*'): 0.0
955 p('6|'*'): 0.0
956 p('7|'*'): 0.0
957 p('8|'*'): 0.0
958 p('9|'*'): 0.0
959 p('a|'*'): 0.0
960 p('b|'*'): 0.0
961 p('c|'*'): 0.0034965034965034965
962 p('d|'*'): 0.0034965034965034965
963 p('e|'*'): 0.0
964 p('f|'*'): 0.0

```

```

965 p('g'|'*'): 0.0
966 p('h'|'*'): 0.0
967 p('i'|'*'): 0.0
968 p('j'|'*'): 0.0
969 p('k'|'*'): 0.0034965034965034965
970 p('l'|'*'): 0.0
971 p('m'|'*'): 0.0
972 p('n'|'*'): 0.0
973 p('o'|'*'): 0.0
974 p('p'|'*'): 0.0034965034965034965
975 p('q'|'*'): 0.0
976 p('r'|'*'): 0.0
977 p('s'|'*'): 0.0
978 p('t'|'*'): 0.0
979 p('u'|'*'): 0.0
980 p('v'|'*'): 0.0
981 p('w'|'*'): 0.0
982 p('x'|'*'): 0.0
983 p('y'|'*'): 0.0
984 p('z'|'*'): 0.0
985
986
987 Symbol: 0
988 Marginal Probability p('0'): 5.331317587540711e-05
989 Transition Probabilities:
990 p('='|'0'): 0.0
991 p(' '|'0'): 0.2619047619047619
992 p('-'|'0'): 0.0
993 p(', '|'0'): 0.06547619047619048
994 p('; '|'0'): 0.0
995 p(':', '|'0'): 0.0
996 p('!'|'0'): 0.0
997 p('? '|'0'): 0.0
998 p('/ '|'0'): 0.0
999 p('. '|'0'): 0.005952380952380952
1000 p('' '|'0'): 0.0
1001 p('"'|'0'): 0.0
1002 p('('|'0'): 0.0
1003 p(')'|'0'): 0.005952380952380952
1004 p('['|'0'): 0.0
1005 p(']'|'0'): 0.005952380952380952
1006 p('*'|'0'): 0.0
1007 p('0'|'0'): 0.16071428571428573
1008 p('1'|'0'): 0.023809523809523808
1009 p('2'|'0'): 0.017857142857142856
1010 p('3'|'0'): 0.0
1011 p('4'|'0'): 0.0

```

```

1012 p('5'|'0'): 0.16071428571428573
1013 p('6'|'0'): 0.07142857142857142
1014 p('7'|'0'): 0.08928571428571429
1015 p('8'|'0'): 0.02976190476190476
1016 p('9'|'0'): 0.10119047619047619
1017 p('a'|'0'): 0.0
1018 p('b'|'0'): 0.0
1019 p('c'|'0'): 0.0
1020 p('d'|'0'): 0.0
1021 p('e'|'0'): 0.0
1022 p('f'|'0'): 0.0
1023 p('g'|'0'): 0.0
1024 p('h'|'0'): 0.0
1025 p('i'|'0'): 0.0
1026 p('j'|'0'): 0.0
1027 p('k'|'0'): 0.0
1028 p('l'|'0'): 0.0
1029 p('m'|'0'): 0.0
1030 p('n'|'0'): 0.0
1031 p('o'|'0'): 0.0
1032 p('p'|'0'): 0.0
1033 p('q'|'0'): 0.0
1034 p('r'|'0'): 0.0
1035 p('s'|'0'): 0.0
1036 p('t'|'0'): 0.0
1037 p('u'|'0'): 0.0
1038 p('v'|'0'): 0.0
1039 p('w'|'0'): 0.0
1040 p('x'|'0'): 0.0
1041 p('y'|'0'): 0.0
1042 p('z'|'0'): 0.0
1043
1044
1045 Symbol: 1
1046 Marginal Probability p('1'): 0.0001237627297107665
1047 Transition Probabilities:
1048 p('= '|'1'): 0.0
1049 p(' '|'1'): 0.046153846153846156
1050 p('- '|'1'): 0.0
1051 p(', '|'1'): 0.010256410256410256
1052 p('; '|'1'): 0.0
1053 p(':', '|'1'): 0.0
1054 p('! '|'1'): 0.0
1055 p('? '|'1'): 0.0
1056 p('/ '|'1'): 0.0
1057 p('. '|'1'): 0.12051282051282051
1058 p('' '|'1'): 0.0

```

```

1059 p('"'|'1'): 0.0
1060 p(' '|'1'): 0.002564102564102564
1061 p(')'|'1'): 0.046153846153846156
1062 p('['|'1'): 0.0
1063 p(']'|'1'): 0.0
1064 p('*'|'1'): 0.0
1065 p('0'|'1'): 0.038461538461538464
1066 p('1'|'1'): 0.041025641025641026
1067 p('2'|'1'): 0.15897435897435896
1068 p('3'|'1'): 0.05128205128205128
1069 p('4'|'1'): 0.005128205128205128
1070 p('5'|'1'): 0.023076923076923078
1071 p('6'|'1'): 0.007692307692307693
1072 p('7'|'1'): 0.020512820512820513
1073 p('8'|'1'): 0.4256410256410256
1074 p('9'|'1'): 0.0
1075 p('a'|'1'): 0.0
1076 p('b'|'1'): 0.0
1077 p('c'|'1'): 0.0
1078 p('d'|'1'): 0.0
1079 p('e'|'1'): 0.0
1080 p('f'|'1'): 0.0
1081 p('g'|'1'): 0.0
1082 p('h'|'1'): 0.0
1083 p('i'|'1'): 0.0
1084 p('j'|'1'): 0.0
1085 p('k'|'1'): 0.0
1086 p('l'|'1'): 0.0
1087 p('m'|'1'): 0.0
1088 p('n'|'1'): 0.0
1089 p('o'|'1'): 0.0
1090 p('p'|'1'): 0.0
1091 p('q'|'1'): 0.0
1092 p('r'|'1'): 0.0
1093 p('s'|'1'): 0.002564102564102564
1094 p('t'|'1'): 0.0
1095 p('u'|'1'): 0.0
1096 p('v'|'1'): 0.0
1097 p('w'|'1'): 0.0
1098 p('x'|'1'): 0.0
1099 p('y'|'1'): 0.0
1100 p('z'|'1'): 0.0
1101
1102
1103 Symbol: 2
1104 Marginal Probability p('2'): 4.474498689543097e-05
1105 Transition Probabilities:

```



```

1106 p('='|'2'): 0.0
1107 p(' '|'2'): 0.2978723404255319
1108 p('-'|'2'): 0.0
1109 p(', '|'2'): 0.09219858156028368
1110 p('; '|'2'): 0.0
1111 p(':', '|'2'): 0.0
1112 p('!'|'2'): 0.0
1113 p('? '|'2'): 0.0070921985815602835
1114 p('/ '|'2'): 0.0
1115 p('. '|'2'): 0.09219858156028368
1116 p(''|'2'): 0.0
1117 p('"'|'2'): 0.0
1118 p('('|'2'): 0.0
1119 p(')'|'2'): 0.2765957446808511
1120 p('['|'2'): 0.0
1121 p(']'|'2'): 0.0070921985815602835
1122 p('*'|'2'): 0.0
1123 p('0'|'2'): 0.0851063829787234
1124 p('1'|'2'): 0.014184397163120567
1125 p('2'|'2'): 0.03546099290780142
1126 p('3'|'2'): 0.014184397163120567
1127 p('4'|'2'): 0.02127659574468085
1128 p('5'|'2'): 0.02127659574468085
1129 p('6'|'2'): 0.014184397163120567
1130 p('7'|'2'): 0.014184397163120567
1131 p('8'|'2'): 0.0
1132 p('9'|'2'): 0.0
1133 p('a'|'2'): 0.0
1134 p('b'|'2'): 0.0
1135 p('c'|'2'): 0.0
1136 p('d'|'2'): 0.0
1137 p('e'|'2'): 0.0
1138 p('f'|'2'): 0.0
1139 p('g'|'2'): 0.0
1140 p('h'|'2'): 0.0
1141 p('i'|'2'): 0.0
1142 p('j'|'2'): 0.0
1143 p('k'|'2'): 0.0
1144 p('l'|'2'): 0.0
1145 p('m'|'2'): 0.0
1146 p('n'|'2'): 0.0070921985815602835
1147 p('o'|'2'): 0.0
1148 p('p'|'2'): 0.0
1149 p('q'|'2'): 0.0
1150 p('r'|'2'): 0.0
1151 p('s'|'2'): 0.0
1152 p('t'|'2'): 0.0

```

```

1153 p('u'|'2'): 0.0
1154 p('v'|'2'): 0.0
1155 p('w'|'2'): 0.0
1156 p('x'|'2'): 0.0
1157 p('y'|'2'): 0.0
1158 p('z'|'2'): 0.0
1159
1160
1161 Symbol: 3
1162 Marginal Probability p('3'): 1.8405739290319122e-05
1163 Transition Probabilities:
1164 p('= '|'3'): 0.0
1165 p(' '|'3'): 0.22413793103448276
1166 p('- '|'3'): 0.0
1167 p(', '|'3'): 0.10344827586206896
1168 p('; '|'3'): 0.0
1169 p(': '|'3'): 0.0
1170 p('! '|'3'): 0.0
1171 p('? '|'3'): 0.0
1172 p('/ '|'3'): 0.0
1173 p('. '|'3'): 0.1206896551724138
1174 p('' '|'3'): 0.0
1175 p('" '|'3'): 0.0
1176 p('(' '|'3'): 0.0
1177 p(')' '|'3'): 0.27586206896551724
1178 p('[' '|'3'): 0.0
1179 p(']' '|'3'): 0.0
1180 p('* '|'3'): 0.0
1181 p('0 '|'3'): 0.1206896551724138
1182 p('1 '|'3'): 0.017241379310344827
1183 p('2 '|'3'): 0.0
1184 p('3 '|'3'): 0.0
1185 p('4 '|'3'): 0.0
1186 p('5 '|'3'): 0.0
1187 p('6 '|'3'): 0.0
1188 p('7 '|'3'): 0.0
1189 p('8 '|'3'): 0.0
1190 p('9 '|'3'): 0.0
1191 p('a '|'3'): 0.0
1192 p('b '|'3'): 0.0
1193 p('c '|'3'): 0.0
1194 p('d '|'3'): 0.0
1195 p('e '|'3'): 0.0
1196 p('f '|'3'): 0.0
1197 p('g '|'3'): 0.0
1198 p('h '|'3'): 0.0
1199 p('i '|'3'): 0.0

```

```

1200 p('j'|'3'): 0.0
1201 p('k'|'3'): 0.0
1202 p('l'|'3'): 0.0
1203 p('m'|'3'): 0.0
1204 p('n'|'3'): 0.0
1205 p('o'|'3'): 0.0
1206 p('p'|'3'): 0.0
1207 p('q'|'3'): 0.0
1208 p('r'|'3'): 0.08620689655172414
1209 p('s'|'3'): 0.0
1210 p('t'|'3'): 0.05172413793103448
1211 p('u'|'3'): 0.0
1212 p('v'|'3'): 0.0
1213 p('w'|'3'): 0.0
1214 p('x'|'3'): 0.0
1215 p('y'|'3'): 0.0
1216 p('z'|'3'): 0.0
1217
1218
1219 Symbol: 4
1220 Marginal Probability p('4'): 7.2988276496093065e-06
1221 Transition Probabilities:
1222 p('= '|'4'): 0.0
1223 p(' '|'4'): 0.17391304347826086
1224 p('- '|'4'): 0.043478260869565216
1225 p(', '|'4'): 0.08695652173913043
1226 p('; '|'4'): 0.0
1227 p(':', '|'4'): 0.0
1228 p('! '|'4'): 0.0
1229 p('? '|'4'): 0.0
1230 p('/ '|'4'): 0.0
1231 p('. '|'4'): 0.2608695652173913
1232 p('' '|'4'): 0.0
1233 p('"' '|'4'): 0.0
1234 p('( '|'4'): 0.0
1235 p(')' '|'4'): 0.13043478260869565
1236 p('[' '|'4'): 0.0
1237 p(']' '|'4'): 0.0
1238 p('* '|'4'): 0.0
1239 p('0 '|'4'): 0.08695652173913043
1240 p('1 '|'4'): 0.08695652173913043
1241 p('2 '|'4'): 0.0
1242 p('3 '|'4'): 0.0
1243 p('4 '|'4'): 0.0
1244 p('5 '|'4'): 0.0
1245 p('6 '|'4'): 0.0
1246 p('7 '|'4'): 0.0

```

```

1247 p('8'|'4'): 0.0
1248 p('9'|'4'): 0.0
1249 p('a'|'4'): 0.0
1250 p('b'|'4'): 0.0
1251 p('c'|'4'): 0.0
1252 p('d'|'4'): 0.0
1253 p('e'|'4'): 0.0
1254 p('f'|'4'): 0.0
1255 p('g'|'4'): 0.0
1256 p('h'|'4'): 0.0
1257 p('i'|'4'): 0.0
1258 p('j'|'4'): 0.0
1259 p('k'|'4'): 0.0
1260 p('l'|'4'): 0.0
1261 p('m'|'4'): 0.0
1262 p('n'|'4'): 0.0
1263 p('o'|'4'): 0.0
1264 p('p'|'4'): 0.0
1265 p('q'|'4'): 0.0
1266 p('r'|'4'): 0.0
1267 p('s'|'4'): 0.0
1268 p('t'|'4'): 0.08695652173913043
1269 p('u'|'4'): 0.0
1270 p('v'|'4'): 0.0
1271 p('w'|'4'): 0.0
1272 p('x'|'4'): 0.043478260869565216
1273 p('y'|'4'): 0.0
1274 p('z'|'4'): 0.0
1275
1276
1277 Symbol: 5
1278 Marginal Probability p('5'): 1.586701662958545e-05
1279 Transition Probabilities:
1280 p('= '|'5'): 0.0
1281 p(' '|'5'): 0.22
1282 p('- '|'5'): 0.02
1283 p(', '|'5'): 0.32
1284 p('; '|'5'): 0.02
1285 p(':', '|'5'): 0.02
1286 p('! '|'5'): 0.0
1287 p('? '|'5'): 0.0
1288 p('/ '|'5'): 0.02
1289 p('. '|'5'): 0.16
1290 p('' '|'5'): 0.0
1291 p('"' '|'5'): 0.0
1292 p('(' '|'5'): 0.0
1293 p(')' '|'5'): 0.02

```

```

1294 p('['|'5'): 0.0
1295 p(']'|'5'): 0.0
1296 p('*'|'5'): 0.0
1297 p('0'|'5'): 0.12
1298 p('1'|'5'): 0.0
1299 p('2'|'5'): 0.0
1300 p('3'|'5'): 0.0
1301 p('4'|'5'): 0.02
1302 p('5'|'5'): 0.0
1303 p('6'|'5'): 0.0
1304 p('7'|'5'): 0.0
1305 p('8'|'5'): 0.0
1306 p('9'|'5'): 0.02
1307 p('a'|'5'): 0.0
1308 p('b'|'5'): 0.0
1309 p('c'|'5'): 0.0
1310 p('d'|'5'): 0.0
1311 p('e'|'5'): 0.0
1312 p('f'|'5'): 0.0
1313 p('g'|'5'): 0.0
1314 p('h'|'5'): 0.0
1315 p('i'|'5'): 0.0
1316 p('j'|'5'): 0.0
1317 p('k'|'5'): 0.0
1318 p('l'|'5'): 0.0
1319 p('m'|'5'): 0.0
1320 p('n'|'5'): 0.0
1321 p('o'|'5'): 0.0
1322 p('p'|'5'): 0.0
1323 p('q'|'5'): 0.0
1324 p('r'|'5'): 0.0
1325 p('s'|'5'): 0.0
1326 p('t'|'5'): 0.02
1327 p('u'|'5'): 0.0
1328 p('v'|'5'): 0.0
1329 p('w'|'5'): 0.0
1330 p('x'|'5'): 0.0
1331 p('y'|'5'): 0.02
1332 p('z'|'5'): 0.0
1333
1334
1335 Symbol: 6
1336 Marginal Probability p('6'): 1.650169729476887e-05
1337 Transition Probabilities:
1338 p('='|'6'): 0.0
1339 p(' '|'6'): 0.21153846153846154
1340 p('-'|'6'): 0.038461538461538464

```

1341 p(', '| '6'): 0.15384615384615385
 1342 p('; '| '6'): 0.019230769230769232
 1343 p(': '| '6'): 0.0
 1344 p('! '| '6'): 0.0
 1345 p('? '| '6'): 0.019230769230769232
 1346 p('/ '| '6'): 0.0
 1347 p('. '| '6'): 0.09615384615384616
 1348 p('' '| '6'): 0.0
 1349 p('"' '| '6'): 0.0
 1350 p('(' '| '6'): 0.0
 1351 p(')' '| '6'): 0.0
 1352 p('['| '6'): 0.0
 1353 p('] '| '6'): 0.0
 1354 p('* '| '6'): 0.0
 1355 p('0 '| '6'): 0.09615384615384616
 1356 p('1 '| '6'): 0.0
 1357 p('2 '| '6'): 0.038461538461538464
 1358 p('3 '| '6'): 0.0
 1359 p('4 '| '6'): 0.019230769230769232
 1360 p('5 '| '6'): 0.0
 1361 p('6 '| '6'): 0.23076923076923078
 1362 p('7 '| '6'): 0.019230769230769232
 1363 p('8 '| '6'): 0.0
 1364 p('9 '| '6'): 0.0
 1365 p('a '| '6'): 0.0
 1366 p('b '| '6'): 0.0
 1367 p('c '| '6'): 0.0
 1368 p('d '| '6'): 0.0
 1369 p('e '| '6'): 0.0
 1370 p('f '| '6'): 0.0
 1371 p('g '| '6'): 0.0
 1372 p('h '| '6'): 0.0
 1373 p('i '| '6'): 0.0
 1374 p('j '| '6'): 0.0
 1375 p('k '| '6'): 0.0
 1376 p('l '| '6'): 0.0
 1377 p('m '| '6'): 0.0
 1378 p('n '| '6'): 0.0
 1379 p('o '| '6'): 0.0
 1380 p('p '| '6'): 0.0
 1381 p('q '| '6'): 0.0
 1382 p('r '| '6'): 0.0
 1383 p('s '| '6'): 0.0
 1384 p('t '| '6'): 0.057692307692307696
 1385 p('u '| '6'): 0.0
 1386 p('v '| '6'): 0.0
 1387 p('w '| '6'): 0.0

```

1388 p('x'|'6'): 0.0
1389 p('y'|'6'): 0.0
1390 p('z'|'6'): 0.0
1391
1392
1393 Symbol: 7
1394 Marginal Probability p('7'): 1.2058932638484941e-05
1395 Transition Probabilities:
1396 p('='|'7'): 0.0
1397 p(' '|'7'): 0.3684210526315789
1398 p('-'|'7'): 0.0
1399 p(', '|'7'): 0.18421052631578946
1400 p('; '|'7'): 0.0
1401 p(':', '|'7'): 0.0
1402 p('!'|'7'): 0.02631578947368421
1403 p('? '|'7'): 0.0
1404 p('/ '|'7'): 0.0
1405 p('. '|'7'): 0.10526315789473684
1406 p(''|'7'): 0.0
1407 p('"'|'7'): 0.0
1408 p('('|'7'): 0.0
1409 p(')'|'7'): 0.0
1410 p('[ '|'7'): 0.0
1411 p(']'|'7'): 0.0
1412 p('* '|'7'): 0.0
1413 p('0'|'7'): 0.07894736842105263
1414 p('1'|'7'): 0.02631578947368421
1415 p('2'|'7'): 0.0
1416 p('3'|'7'): 0.0
1417 p('4'|'7'): 0.0
1418 p('5'|'7'): 0.0
1419 p('6'|'7'): 0.0
1420 p('7'|'7'): 0.0
1421 p('8'|'7'): 0.10526315789473684
1422 p('9'|'7'): 0.02631578947368421
1423 p('a'|'7'): 0.0
1424 p('b'|'7'): 0.0
1425 p('c'|'7'): 0.0
1426 p('d'|'7'): 0.0
1427 p('e'|'7'): 0.0
1428 p('f'|'7'): 0.0
1429 p('g'|'7'): 0.0
1430 p('h'|'7'): 0.0
1431 p('i'|'7'): 0.0
1432 p('j'|'7'): 0.0
1433 p('k'|'7'): 0.0
1434 p('l'|'7'): 0.0

```

```

1435 p('m'|'7'): 0.0
1436 p('n'|'7'): 0.0
1437 p('o'|'7'): 0.0
1438 p('p'|'7'): 0.0
1439 p('q'|'7'): 0.0
1440 p('r'|'7'): 0.0
1441 p('s'|'7'): 0.0
1442 p('t'|'7'): 0.07894736842105263
1443 p('u'|'7'): 0.0
1444 p('v'|'7'): 0.0
1445 p('w'|'7'): 0.0
1446 p('x'|'7'): 0.0
1447 p('y'|'7'): 0.0
1448 p('z'|'7'): 0.0
1449
1450
1451 Symbol: 8
1452 Marginal Probability p('8'): 6.0929343857608124e-05
1453 Transition Probabilities:
1454 p('= '| '8'): 0.0
1455 p(' '| '8'): 0.046875
1456 p('- '| '8'): 0.0
1457 p(', '| '8'): 0.010416666666666666
1458 p('; '| '8'): 0.0
1459 p(': '| '8'): 0.0
1460 p('! '| '8'): 0.0
1461 p('? '| '8'): 0.0
1462 p('/ '| '8'): 0.0
1463 p('. '| '8'): 0.020833333333333332
1464 p('' '| '8'): 0.0
1465 p('"' '| '8'): 0.0
1466 p('( '| '8'): 0.0
1467 p(')' '| '8'): 0.0
1468 p('[ '| '8'): 0.0
1469 p('] '| '8'): 0.0
1470 p('* '| '8'): 0.0
1471 p('0 '| '8'): 0.4114583333333333
1472 p('1 '| '8'): 0.4427083333333333
1473 p('2 '| '8'): 0.020833333333333332
1474 p('3 '| '8'): 0.0
1475 p('4 '| '8'): 0.005208333333333333
1476 p('5 '| '8'): 0.0
1477 p('6 '| '8'): 0.005208333333333333
1478 p('7 '| '8'): 0.005208333333333333
1479 p('8 '| '8'): 0.005208333333333333
1480 p('9 '| '8'): 0.015625
1481 p('a '| '8'): 0.0

```



```

1482 p('b'|'8'): 0.0
1483 p('c'|'8'): 0.0
1484 p('d'|'8'): 0.0
1485 p('e'|'8'): 0.0
1486 p('f'|'8'): 0.0
1487 p('g'|'8'): 0.0
1488 p('h'|'8'): 0.0
1489 p('i'|'8'): 0.0
1490 p('j'|'8'): 0.0
1491 p('k'|'8'): 0.0
1492 p('l'|'8'): 0.0
1493 p('m'|'8'): 0.0
1494 p('n'|'8'): 0.0
1495 p('o'|'8'): 0.0
1496 p('p'|'8'): 0.0
1497 p('q'|'8'): 0.0
1498 p('r'|'8'): 0.0
1499 p('s'|'8'): 0.0
1500 p('t'|'8'): 0.010416666666666666
1501 p('u'|'8'): 0.0
1502 p('v'|'8'): 0.0
1503 p('w'|'8'): 0.0
1504 p('x'|'8'): 0.0
1505 p('y'|'8'): 0.0
1506 p('z'|'8'): 0.0
1507
1508
1509 Symbol: 9
1510 Marginal Probability p('9'): 9.837550310342979e-06
1511 Transition Probabilities:
1512 p('='|'9'): 0.0
1513 p(' '|'9'): 0.45161290322580644
1514 p('-'|'9'): 0.0
1515 p(', '|'9'): 0.16129032258064516
1516 p('; '|'9'): 0.0
1517 p(':', '|'9'): 0.03225806451612903
1518 p('!'|'9'): 0.0
1519 p('? '|'9'): 0.0
1520 p('/ '|'9'): 0.0
1521 p('.', '|'9'): 0.16129032258064516
1522 p(''|'9'): 0.0
1523 p('"'|'9'): 0.0
1524 p('('|'9'): 0.0
1525 p(')'|'9'): 0.0
1526 p('['|'9'): 0.0
1527 p(']'|'9'): 0.0
1528 p('*'|'9'): 0.0

```

```

1529 p('0'|'9'): 0.0967741935483871
1530 p('1'|'9'): 0.0
1531 p('2'|'9'): 0.0
1532 p('3'|'9'): 0.0
1533 p('4'|'9'): 0.0
1534 p('5'|'9'): 0.0
1535 p('6'|'9'): 0.03225806451612903
1536 p('7'|'9'): 0.03225806451612903
1537 p('8'|'9'): 0.0
1538 p('9'|'9'): 0.0
1539 p('a'|'9'): 0.0
1540 p('b'|'9'): 0.0
1541 p('c'|'9'): 0.0
1542 p('d'|'9'): 0.0
1543 p('e'|'9'): 0.0
1544 p('f'|'9'): 0.0
1545 p('g'|'9'): 0.0
1546 p('h'|'9'): 0.0
1547 p('i'|'9'): 0.0
1548 p('j'|'9'): 0.0
1549 p('k'|'9'): 0.0
1550 p('l'|'9'): 0.0
1551 p('m'|'9'): 0.0
1552 p('n'|'9'): 0.0
1553 p('o'|'9'): 0.0
1554 p('p'|'9'): 0.0
1555 p('q'|'9'): 0.0
1556 p('r'|'9'): 0.0
1557 p('s'|'9'): 0.0
1558 p('t'|'9'): 0.03225806451612903
1559 p('u'|'9'): 0.0
1560 p('v'|'9'): 0.0
1561 p('w'|'9'): 0.0
1562 p('x'|'9'): 0.0
1563 p('y'|'9'): 0.0
1564 p('z'|'9'): 0.0
1565
1566
1567 Symbol: a
1568 Marginal Probability p('a'): 0.06418525566999905
1569 Transition Probabilities:
1570 p('='|'a'): 0.0
1571 p(' '|'a'): 0.06588549391871848
1572 p('-'|'a'): 0.000182932858696727
1573 p(', '|'a'): 0.004677148225056858
1574 p('; '|'a'): 9.393849500642738e-05
1575 p(':', '|'a'): 2.966478789676654e-05

```

1576 p('!'|'a'): 0.0006377929397804805
 1577 p('?'|'a'): 0.00026698309107089886
 1578 p('/'|'a'): 0.0
 1579 p('.'|'a'): 0.0025116187085929003
 1580 p(''|'a'): 0.0
 1581 p('"'|'a'): 0.0
 1582 p('('|'a'): 0.0
 1583 p(')'|'a'): 4.449718184514981e-05
 1584 p('['|'a'): 0.0
 1585 p(']'|'a'): 0.0
 1586 p('*'|'a'): 0.0
 1587 p('0'|'a'): 0.0
 1588 p('1'|'a'): 0.0
 1589 p('2'|'a'): 0.0
 1590 p('3'|'a'): 0.0
 1591 p('4'|'a'): 0.0
 1592 p('5'|'a'): 0.0
 1593 p('6'|'a'): 0.0
 1594 p('7'|'a'): 0.0
 1595 p('8'|'a'): 0.0
 1596 p('9'|'a'): 0.0
 1597 p('a'|'a'): 3.4608919212894296e-05
 1598 p('b'|'a'): 0.01710175022248591
 1599 p('c'|'a'): 0.034673192920003953
 1600 p('d'|'a'): 0.05567586275091466
 1601 p('e'|'a'): 0.00033620092949668745
 1602 p('f'|'a'): 0.008227034510036587
 1603 p('g'|'a'): 0.016691387323247306
 1604 p('h'|'a'): 0.0015524572332641154
 1605 p('i'|'a'): 0.04310293681400178
 1606 p('j'|'a'): 0.0008306140611094631
 1607 p('k'|'a'): 0.012523484623751607
 1608 p('l'|'a'): 0.07027588252743992
 1609 p('m'|'a'): 0.022733115791555424
 1610 p('n'|'a'): 0.2246366063482646
 1611 p('o'|'a'): 0.00020765351527736578
 1612 p('p'|'a'): 0.02289627212498764
 1613 p('q'|'a'): 9.888262632255513e-06
 1614 p('r'|'a'): 0.08741718580045486
 1615 p('s'|'a'): 0.09649461089686542
 1616 p('t'|'a'): 0.13835657075051913
 1617 p('u'|'a'): 0.012414713734796795
 1618 p('v'|'a'): 0.021106496588549392
 1619 p('w'|'a'): 0.010669435380203698
 1620 p('x'|'a'): 0.00025215069712251556
 1621 p('y'|'a'): 0.0257391476317611
 1622 p('z'|'a'): 0.0017106694353802036

```

1623
1624
1625 Symbol: b
1626 Marginal Probability p('b'): 0.010967599234702054
1627 Transition Probabilities:
1628 p('='|'b'): 0.0
1629 p(' '| 'b'): 0.003414253059807297
1630 p('-'|'b'): 0.00014467173982234312
1631 p(', '| 'b'): 0.0010127021787564017
1632 p('; '| 'b'): 0.0
1633 p(':', '| 'b'): 0.0
1634 p('!'|'b'): 2.893434796446862e-05
1635 p('? '| 'b'): 5.786869592893724e-05
1636 p('/ '| 'b'): 0.0
1637 p('.')|'b'): 0.0005497526113249038
1638 p(''| 'b'): 0.0
1639 p('"'|'b'): 0.0
1640 p('('|'b'): 0.0
1641 p(')'|'b'): 5.786869592893724e-05
1642 p('['|'b'): 0.0
1643 p(']'|'b'): 0.0
1644 p('* '| 'b'): 0.0
1645 p('0'|'b'): 0.0
1646 p('1'|'b'): 0.0
1647 p('2'|'b'): 0.0
1648 p('3'|'b'): 0.0
1649 p('4'|'b'): 0.0
1650 p('5'|'b'): 0.0
1651 p('6'|'b'): 0.0
1652 p('7'|'b'): 0.0
1653 p('8'|'b'): 0.0
1654 p('9'|'b'): 0.0
1655 p('a'|'b'): 0.09224270131072597
1656 p('b'|'b'): 0.006625965683863314
1657 p('c'|'b'): 8.680304389340587e-05
1658 p('d'|'b'): 0.00046294956743149794
1659 p('e'|'b'): 0.3286363241804346
1660 p('f'|'b'): 0.0
1661 p('g'|'b'): 0.0
1662 p('h'|'b'): 2.893434796446862e-05
1663 p('i'|'b'): 0.029079019704290963
1664 p('j'|'b'): 0.005063510893782009
1665 p('k'|'b'): 0.0
1666 p('l'|'b'): 0.10494488006712768
1667 p('m'|'b'): 0.001996470009548335
1668 p('n'|'b'): 0.00023147478371574897
1669 p('o'|'b'): 0.11831254882671219

```

```

1670 p('p'|'b'): 0.0
1671 p('q'|'b'): 0.0
1672 p('r'|'b'): 0.05723214027371893
1673 p('s'|'b'): 0.013888487022944937
1674 p('t'|'b'): 0.007436127426868435
1675 p('u'|'b'): 0.15262868551257197
1676 p('v'|'b'): 0.0013309800063655566
1677 p('w'|'b'): 0.0005786869592893725
1678 p('x'|'b'): 0.0
1679 p('y'|'b'): 0.07392725904921732
1680 p('z'|'b'): 0.0
1681
1682
1683 Symbol: c
1684 Marginal Probability p('c'): 0.019547529806984088
1685 Transition Probabilities:
1686 p('= '| 'c'): 0.0
1687 p(' '| 'c'): 0.008474301113672521
1688 p('- '| 'c'): 3.2468586642423456e-05
1689 p(', '| 'c'): 0.0006169031462060457
1690 p('; '| 'c'): 4.8702879963635184e-05
1691 p(': '| 'c'): 1.6234293321211728e-05
1692 p('! '| 'c'): 4.8702879963635184e-05
1693 p('? '| 'c'): 1.6234293321211728e-05
1694 p('/ '| 'c'): 0.0
1695 p('. '| 'c'): 0.00048702879963635184
1696 p('' '| 'c'): 0.0
1697 p('"' '| 'c'): 0.0
1698 p('( '| 'c'): 0.0
1699 p(')' '| 'c'): 4.8702879963635184e-05
1700 p('[ '| 'c'): 0.0
1701 p(']' '| 'c'): 0.0
1702 p('* '| 'c'): 0.0
1703 p('0 '| 'c'): 0.0
1704 p('1 '| 'c'): 0.0
1705 p('2 '| 'c'): 0.0
1706 p('3 '| 'c'): 0.0
1707 p('4 '| 'c'): 0.0
1708 p('5 '| 'c'): 0.0
1709 p('6 '| 'c'): 0.0
1710 p('7 '| 'c'): 0.0
1711 p('8 '| 'c'): 0.0
1712 p('9 '| 'c'): 0.0
1713 p('a '| 'c'): 0.10912691970518523
1714 p('b '| 'c'): 0.0
1715 p('c '| 'c'): 0.017029773693951103
1716 p('d '| 'c'): 0.00014610863989090555

```

```

1717 p('e'|'c'): 0.21315627130751
1718 p('f'|'c'): 0.0
1719 p('g'|'c'): 0.0
1720 p('h'|'c'): 0.18843144257930453
1721 p('i'|'c'): 0.04173836812883535
1722 p('j'|'c'): 0.0
1723 p('k'|'c'): 0.042631254261501994
1724 p('l'|'c'): 0.03254975810902951
1725 p('m'|'c'): 3.2468586642423456e-05
1726 p('n'|'c'): 0.0
1727 p('o'|'c'): 0.20841585765771617
1728 p('p'|'c'): 0.0
1729 p('q'|'c'): 0.0018994123185817721
1730 p('r'|'c'): 0.03362122146822949
1731 p('s'|'c'): 0.001558492158836326
1732 p('t'|'c'): 0.06720997434981656
1733 p('u'|'c'): 0.02743595571284782
1734 p('v'|'c'): 0.0
1735 p('w'|'c'): 0.0001136400532484821
1736 p('x'|'c'): 0.0
1737 p('y'|'c'): 0.005032630929575636
1738 p('z'|'c'): 8.117146660605864e-05
1739
1740
1741 Symbol: d
1742 Marginal Probability p('d'): 0.03730081737349466
1743 Transition Probabilities:
1744 p('='|'d'): 0.0
1745 p(' '|'d'): 0.5782188494325433
1746 p('-'|'d'): 0.0018971941944156132
1747 p(', '|'d'): 0.035646832621531024
1748 p('; '|'d'): 0.0010038964795562437
1749 p(':', '|'d'): 0.0015824130948937402
1750 p('!'|'d'): 0.002688400741862483
1751 p('? '|'d'): 0.0015824130948937402
1752 p('/ '|'d'): 0.0
1753 p('. '|'d'): 0.0240084395365061
1754 p('' '|'d'): 0.0
1755 p('"'|'d'): 8.507597284374947e-06
1756 p('('|'d'): 0.0
1757 p(')'|'d'): 0.0003998570723656225
1758 p('['|'d'): 0.0
1759 p(']'|'d'): 0.0
1760 p('*'|'d'): 0.0
1761 p('0'|'d'): 0.0
1762 p('1'|'d'): 0.0
1763 p('2'|'d'): 0.0

```

```

1764 p('3'|'d'): 0.0
1765 p('4'|'d'): 0.0
1766 p('5'|'d'): 0.0
1767 p('6'|'d'): 0.0
1768 p('7'|'d'): 0.0
1769 p('8'|'d'): 0.0
1770 p('9'|'d'): 0.0
1771 p('a'|'d'): 0.020750029776590496
1772 p('b'|'d'): 0.00010209116741249936
1773 p('c'|'d'): 8.507597284374947e-05
1774 p('d'|'d'): 0.011085399261540556
1775 p('e'|'d'): 0.11691990947916489
1776 p('f'|'d'): 0.0007231457691718705
1777 p('g'|'d'): 0.003692297221418727
1778 p('h'|'d'): 0.0004253798642187473
1779 p('i'|'d'): 0.06630821323441834
1780 p('j'|'d'): 0.0020077929591124873
1781 p('k'|'d'): 0.0005870242126218713
1782 p('l'|'d'): 0.01054091303534056
1783 p('m'|'d'): 0.0026713855472937334
1784 p('n'|'d'): 0.0024672032124687347
1785 p('o'|'d'): 0.04194245461196849
1786 p('p'|'d'): 3.403038913749979e-05
1787 p('q'|'d'): 0.0003828418777968726
1788 p('r'|'d'): 0.030397645097071685
1789 p('s'|'d'): 0.018946419152303006
1790 p('t'|'d'): 0.0003998570723656225
1791 p('u'|'d'): 0.009366864610096816
1792 p('v'|'d'): 0.002722431130999983
1793 p('w'|'d'): 0.00037433428051249765
1794 p('x'|'d'): 0.0
1795 p('y'|'d'): 0.010013442003709313
1796 p('z'|'d'): 1.7015194568749894e-05
1797
1798
1799 Symbol: e
1800 Marginal Probability p('e'): 0.09907745991912265
1801 Transition Probabilities:
1802 p('='|'e'): 0.0
1803 p(' '|'e'): 0.31652210677360254
1804 p('-'|'e'): 0.0009000294671569318
1805 p(', '|'e'): 0.019499570804453383
1806 p('; '|'e'): 0.0005092693426261642
1807 p(': '|'e'): 0.0005060663907857482
1808 p('! '|'e'): 0.0021716013478021343
1809 p('? '|'e'): 0.0018449002600796894
1810 p('/ '|'e'): 0.0

```

```

1811 p('.'|'e'): 0.01401932020550139
1812 p(''|'e'): 3.2029518404161277e-06
1813 p('"'|'e'): 0.0
1814 p('('|'e'): 0.0
1815 p(')'|'e'): 0.0002338154843503773
1816 p('['|'e'): 0.0
1817 p(']'|'e'): 0.0
1818 p('*'|'e'): 0.0
1819 p('0'|'e'): 0.0
1820 p('1'|'e'): 0.0
1821 p('2'|'e'): 0.0
1822 p('3'|'e'): 0.0
1823 p('4'|'e'): 0.0
1824 p('5'|'e'): 0.0
1825 p('6'|'e'): 0.0
1826 p('7'|'e'): 0.0
1827 p('8'|'e'): 0.0
1828 p('9'|'e'): 0.0
1829 p('a'|'e'): 0.04271456574378948
1830 p('b'|'e'): 0.0008135497674656964
1831 p('c'|'e'): 0.017215866142236684
1832 p('d'|'e'): 0.09163324920246499
1833 p('e'|'e'): 0.02545065532394655
1834 p('f'|'e'): 0.010143748478597876
1835 p('g'|'e'): 0.006860722842171345
1836 p('h'|'e'): 0.002286907614057115
1837 p('i'|'e'): 0.011101431078882298
1838 p('j'|'e'): 0.00027225090643537086
1839 p('k'|'e'): 0.0008583910932315221
1840 p('l'|'e'): 0.03253878774678744
1841 p('m'|'e'): 0.022750566922475755
1842 p('n'|'e'): 0.08393335297810463
1843 p('o'|'e'): 0.005163158366750797
1844 p('p'|'e'): 0.010976515957106069
1845 p('q'|'e'): 0.0008167527193061125
1846 p('r'|'e'): 0.14110924628137292
1847 p('s'|'e'): 0.06281308854240068
1848 p('t'|'e'): 0.02367942295619643
1849 p('u'|'e'): 0.0009961180223694156
1850 p('v'|'e'): 0.016985253609726723
1851 p('w'|'e'): 0.010191792756204118
1852 p('x'|'e'): 0.010038051067864144
1853 p('y'|'e'): 0.01190216903898633
1854 p('z'|'e'): 0.0005445018128707417
1855
1856
1857 Symbol: f

```



```

1858 Marginal Probability p('f'): 0.01738580746136937
1859 Transition Probabilities:
1860 p('='|'f'): 0.0
1861 p(' '|'f'): 0.3353776512247654
1862 p('-'|'f'): 0.0030664768371481766
1863 p(', '|'f'): 0.009071660643230022
1864 p('; '|'f'): 0.00016427554484722376
1865 p(':', '|'f'): 0.0003468039280108057
1866 p('!'|'f'): 0.0009491475924506261
1867 p('? '|'f'): 0.0005475851494907458
1868 p('/ '|'f'): 0.0
1869 p('. '|'f'): 0.007611433577921367
1870 p('' '|'f'): 0.0
1871 p('"'|'f'): 0.0
1872 p('('|'f'): 0.0
1873 p(')'|'f'): 0.00021903405979629831
1874 p('['|'f'): 0.0
1875 p(']'|'f'): 0.0
1876 p('* '|'f'): 0.0
1877 p('0 '|'f'): 0.0
1878 p('1 '|'f'): 0.0
1879 p('2 '|'f'): 0.0
1880 p('3 '|'f'): 0.0
1881 p('4 '|'f'): 0.0
1882 p('5 '|'f'): 0.0
1883 p('6 '|'f'): 0.0
1884 p('7 '|'f'): 0.0
1885 p('8 '|'f'): 0.0
1886 p('9 '|'f'): 0.0
1887 p('a '|'f'): 0.07032818603292812
1888 p('b '|'f'): 0.0004928266345416712
1889 p('c '|'f'): 0.0
1890 p('d '|'f'): 0.0
1891 p('e '|'f'): 0.08359799948892052
1892 p('f '|'f'): 0.05172854378855912
1893 p('g '|'f'): 0.0
1894 p('h '|'f'): 5.475851494907458e-05
1895 p('i '|'f'): 0.08514949074581098
1896 p('j '|'f'): 0.0
1897 p('k '|'f'): 0.00012776986821450736
1898 p('l '|'f'): 0.022067681524477058
1899 p('m '|'f'): 3.6505676632716386e-05
1900 p('n '|'f'): 9.126419158179097e-05
1901 p('o '|'f'): 0.14768371481765413
1902 p('p '|'f'): 0.0
1903 p('q '|'f'): 0.0
1904 p('r '|'f'): 0.10331106487058737

```

```

1905 p('s'|'f'): 0.0027196729091373707
1906 p('t'|'f'): 0.03924360238017011
1907 p('u'|'f'): 0.0339685321067426
1908 p('v'|'f'): 0.0
1909 p('w'|'f'): 0.0004928266345416712
1910 p('x'|'f'): 0.0
1911 p('y'|'f'): 0.0015514912568904465
1912 p('z'|'f'): 0.0
1913
1914
1915 Symbol: g
1916 Marginal Probability p('g'): 0.016216725676101513
1917 Transition Probabilities:
1918 p('= '| 'g'): 0.0
1919 p(' '| 'g'): 0.3759344056984071
1920 p('- '| 'g'): 0.0020155766897577395
1921 p(', '| 'g'): 0.028668153888301826
1922 p('; '| 'g'): 0.0008414543462095417
1923 p(': '| 'g'): 0.0011741223435481978
1924 p('! '| 'g'): 0.0026809126844350513
1925 p('? '| 'g'): 0.0026809126844350513
1926 p('/ '| 'g'): 7.827482290321318e-05
1927 p('. '| 'g'): 0.020625415834996674
1928 p('' '| 'g'): 0.0
1929 p('"' '| 'g'): 9.784352862901647e-05
1930 p('( '| 'g'): 0.0
1931 p(')' '| 'g'): 0.0004109428202418692
1932 p('[ '| 'g'): 0.0
1933 p('] '| 'g'): 0.0
1934 p('* '| 'g'): 0.0
1935 p('0 '| 'g'): 0.0
1936 p('1 '| 'g'): 0.0
1937 p('2 '| 'g'): 0.0
1938 p('3 '| 'g'): 0.0
1939 p('4 '| 'g'): 0.0
1940 p('5 '| 'g'): 0.0
1941 p('6 '| 'g'): 0.0
1942 p('7 '| 'g'): 0.0
1943 p('8 '| 'g'): 0.0
1944 p('9 '| 'g'): 0.0
1945 p('a '| 'g'): 0.06573128253297326
1946 p('b '| 'g'): 0.0
1947 p('c '| 'g'): 0.0
1948 p('d '| 'g'): 0.0008218856404837384
1949 p('e '| 'g'): 0.11408555438143321
1950 p('f '| 'g'): 0.0
1951 p('g '| 'g'): 0.008708074047982467

```

```

1952 p('h'|'g'): 0.11664905483151344
1953 p('i'|'g'): 0.04882392078587922
1954 p('j'|'g'): 0.0
1955 p('k'|'g'): 0.0
1956 p('l'|'g'): 0.028805134828382453
1957 p('m'|'g'): 0.0003522367030644593
1958 p('n'|'g'): 0.018120621502093853
1959 p('o'|'g'): 0.06034988845837736
1960 p('p'|'g'): 0.0
1961 p('q'|'g'): 0.0
1962 p('r'|'g'): 0.05363782239442683
1963 p('s'|'g'): 0.016868224335642442
1964 p('t'|'g'): 0.00332667997338656
1965 p('u'|'g'): 0.02626120308402802
1966 p('v'|'g'): 0.0
1967 p('w'|'g'): 0.0003522367030644593
1968 p('x'|'g'): 0.0
1969 p('y'|'g'): 0.0018590270439513132
1970 p('z'|'g'): 3.913741145160659e-05
1971
1972
1973 Symbol: h
1974 Marginal Probability p('h'): 0.05282066367922478
1975 Transition Probabilities:
1976 p('= '| 'h'): 0.0
1977 p(' '| 'h'): 0.07947226761511103
1978 p('- '| 'h'): 0.0002222916466403922
1979 p(', '| 'h'): 0.007636018456214554
1980 p('; '| 'h'): 0.00018023647024896665
1981 p(': '| 'h'): 8.41103527828511e-05
1982 p('! '| 'h'): 0.0011775449389599154
1983 p('? '| 'h'): 0.0006608670575795443
1984 p('/ '| 'h'): 0.0
1985 p('. '| 'h'): 0.0032923195232144576
1986 p('' '| 'h'): 0.0
1987 p('"' '| 'h'): 0.0
1988 p('( '| 'h'): 0.0
1989 p(') '| 'h'): 6.608670575795444e-05
1990 p('[ '| 'h'): 0.0
1991 p('] '| 'h'): 0.0
1992 p('* '| 'h'): 0.0
1993 p('0 '| 'h'): 0.0
1994 p('1 '| 'h'): 0.0
1995 p('2 '| 'h'): 0.0
1996 p('3 '| 'h'): 0.0
1997 p('4 '| 'h'): 0.0
1998 p('5 '| 'h'): 0.0

```

```

1999 p('6'|'h'): 0.0
2000 p('7'|'h'): 0.0
2001 p('8'|'h'): 0.0
2002 p('9'|'h'): 0.0
2003 p('a'|'h'): 0.1649223781601461
2004 p('b'|'h'): 0.00037248870518119775
2005 p('c'|'h'): 0.00037248870518119775
2006 p('d'|'h'): 0.0003604729404979333
2007 p('e'|'h'): 0.4499603479765452
2008 p('f'|'h'): 0.00040853599923099105
2009 p('g'|'h'): 6.007882341632221e-06
2010 p('h'|'h'): 3.6047294049793326e-05
2011 p('i'|'h'): 0.15511751417860234
2012 p('j'|'h'): 0.0
2013 p('k'|'h'): 0.000570748822455061
2014 p('l'|'h'): 0.0009011823512448332
2015 p('m'|'h'): 0.0022649716427953475
2016 p('n'|'h'): 0.0006969143516293377
2017 p('o'|'h'): 0.08218783043352879
2018 p('p'|'h'): 2.4031529366528884e-05
2019 p('q'|'h'): 0.0
2020 p('r'|'h'): 0.007696097279630876
2021 p('s'|'h'): 0.0009432375276362588
2022 p('t'|'h'): 0.025191050658463904
2023 p('u'|'h'): 0.009570556570220129
2024 p('v'|'h'): 2.4031529366528884e-05
2025 p('w'|'h'): 0.00021628376429875997
2026 p('x'|'h'): 0.0
2027 p('y'|'h'): 0.0053650389310775735
2028 p('z'|'h'): 0.0
2029
2030
2031 Symbol: i
2032 Marginal Probability p('i'): 0.05444830224508765
2033 Transition Probabilities:
2034 p('='|'i'): 0.0
2035 p(' '|'i'): 0.02912977846681082
2036 p('-'|'i'): 0.00012822231417963946
2037 p(', '|'i'): 0.0012938797158127254
2038 p('; '|'i'): 1.748486102449629e-05
2039 p(':', '|'i'): 5.82828700816543e-06
2040 p('!'|'i'): 0.00012239402717147404
2041 p('? '|'i'): 0.00024478805434294807
2042 p('/ '|'i'): 0.0
2043 p('. '|'i'): 0.0007518490240533404
2044 p(''|'i'): 0.0
2045 p('"'|'i'): 1.165657401633086e-05

```

```

2046 p(''|'i'): 0.0
2047 p(')'|'i'): 0.0
2048 p('['|'i'): 0.0
2049 p(']'|'i'): 0.0
2050 p('*'|'i'): 0.0
2051 p('0'|'i'): 0.0
2052 p('1'|'i'): 0.0
2053 p('2'|'i'): 0.0
2054 p('3'|'i'): 0.0
2055 p('4'|'i'): 0.0
2056 p('5'|'i'): 0.0
2057 p('6'|'i'): 0.0
2058 p('7'|'i'): 0.0
2059 p('8'|'i'): 0.0
2060 p('9'|'i'): 0.0
2061 p('a'|'i'): 0.016954486906753236
2062 p('b'|'i'): 0.007413581074386427
2063 p('c'|'i'): 0.04960455072649598
2064 p('d'|'i'): 0.050385541185590146
2065 p('e'|'i'): 0.04920239892293256
2066 p('f'|'i'): 0.021832763132587703
2067 p('g'|'i'): 0.02506163413511135
2068 p('h'|'i'): 3.496972204899258e-05
2069 p('i'|'i'): 0.0026810120237560978
2070 p('j'|'i'): 0.0
2071 p('k'|'i'): 0.006982287835782185
2072 p('l'|'i'): 0.04617168967868654
2073 p('m'|'i'): 0.05719880869813553
2074 p('n'|'i'): 0.2806611608782063
2075 p('o'|'i'): 0.04237747483637084
2076 p('p'|'i'): 0.004843306503785472
2077 p('q'|'i'): 0.0001573637492204666
2078 p('r'|'i'): 0.03445100450526586
2079 p('s'|'i'): 0.12304096702938039
2080 p('t'|'i'): 0.12318667420458453
2081 p('u'|'i'): 0.0007576773110615059
2082 p('v'|'i'): 0.02002016587304825
2083 p('w'|'i'): 2.331314803266172e-05
2084 p('x'|'i'): 0.0018883649906455994
2085 p('y'|'i'): 5.82828700816543e-06
2086 p('z'|'i'): 0.0033570933167032875
2087
2088
2089 Symbol: j
2090 Marginal Probability p('j'): 0.0008158819950932838
2091 Transition Probabilities:
2092 p('='|'j'): 0.0

```

2093 p(' '|j'): 0.0
 2094 p('-'|j'): 0.0
 2095 p(', '|j'): 0.0
 2096 p('; '|j'): 0.0
 2097 p(':', '|j'): 0.0
 2098 p('! '|j'): 0.0
 2099 p('? '|j'): 0.0
 2100 p('/ '|j'): 0.0
 2101 p('. '|j'): 0.0
 2102 p('' '|j'): 0.0
 2103 p('" '|j'): 0.0
 2104 p('(' '|j'): 0.0
 2105 p(')' '|j'): 0.0
 2106 p('[' '|j'): 0.0
 2107 p(']' '|j'): 0.0
 2108 p('* '|j'): 0.0
 2109 p('0 '|j'): 0.0
 2110 p('1 '|j'): 0.0
 2111 p('2 '|j'): 0.0
 2112 p('3 '|j'): 0.0
 2113 p('4 '|j'): 0.0
 2114 p('5 '|j'): 0.0
 2115 p('6 '|j'): 0.0
 2116 p('7 '|j'): 0.0
 2117 p('8 '|j'): 0.0
 2118 p('9 '|j'): 0.0
 2119 p('a '|j'): 0.03695060287825749
 2120 p('b '|j'): 0.0
 2121 p('c '|j'): 0.0
 2122 p('d '|j'): 0.0
 2123 p('e '|j'): 0.21975884869700504
 2124 p('f '|j'): 0.0
 2125 p('g '|j'): 0.0
 2126 p('h '|j'): 0.0
 2127 p('i '|j'): 0.004278490859587709
 2128 p('j '|j'): 0.0
 2129 p('k '|j'): 0.0
 2130 p('l '|j'): 0.0
 2131 p('m '|j'): 0.0
 2132 p('n '|j'): 0.0
 2133 p('o '|j'): 0.2784908595877091
 2134 p('p '|j'): 0.0
 2135 p('q '|j'): 0.0
 2136 p('r '|j'): 0.0
 2137 p('s '|j'): 0.0
 2138 p('t '|j'): 0.0
 2139 p('u '|j'): 0.4605211979774407

```

2140 p('v'|'j'): 0.0
2141 p('w'|'j'): 0.0
2142 p('x'|'j'): 0.0
2143 p('y'|'j'): 0.0
2144 p('z'|'j'): 0.0
2145
2146
2147 Symbol: k
2148 Marginal Probability p('k'): 0.006325544849550535
2149 Transition Probabilities:
2150 p('=|'k'): 0.0
2151 p(' '|'k'): 0.19906687402799378
2152 p('-|'k'): 0.0016053780163547886
2153 p(',|'k'): 0.027843274971153364
2154 p(';|'k'): 0.0008026890081773943
2155 p(':',|'k'): 0.0005016806301108714
2156 p('!|'k'): 0.003060251843676316
2157 p('?|'k'): 0.0026087392765765313
2158 p('/|'k'): 5.016806301108714e-05
2159 p('.|'k'): 0.019565544574323985
2160 p(''|'k'): 0.0
2161 p('"'|'k'): 0.0
2162 p('('|'k'): 0.0
2163 p(')|'k'): 0.0003010083780665229
2164 p('['|'k'): 0.0
2165 p(']|'k'): 0.0
2166 p('*|'k'): 0.0
2167 p('0'|'k'): 0.0
2168 p('1'|'k'): 0.0
2169 p('2'|'k'): 0.0
2170 p('3'|'k'): 0.0
2171 p('4'|'k'): 0.0
2172 p('5'|'k'): 0.0
2173 p('6'|'k'): 0.0
2174 p('7'|'k'): 0.0
2175 p('8'|'k'): 0.0
2176 p('9'|'k'): 0.0
2177 p('a'|'k'): 0.025234535694576832
2178 p('b'|'k'): 0.00035117644107761
2179 p('c'|'k'): 0.0023578989615210956
2180 p('d'|'k'): 0.00015050418903326144
2181 p('e'|'k'): 0.2836502282646867
2182 p('f'|'k'): 0.0006521848191441328
2183 p('g'|'k'): 0.0006020167561330457
2184 p('h'|'k'): 0.040987307480058195
2185 p('i'|'k'): 0.16981889329252997
2186 p('j'|'k'): 0.0

```

```

2187 p('k'|'k'): 0.0
2188 p('l'|'k'): 0.017006973360758543
2189 p('m'|'k'): 0.001454873827321527
2190 p('n'|'k'): 0.10389805849596147
2191 p('o'|'k'): 0.012792856067827221
2192 p('p'|'k'): 5.016806301108714e-05
2193 p('q'|'k'): 0.0
2194 p('r'|'k'): 0.004765965986053279
2195 p('s'|'k'): 0.03742537500627101
2196 p('t'|'k'): 0.0002508403150554357
2197 p('u'|'k'): 0.03341192996538404
2198 p('v'|'k'): 0.0006020167561330457
2199 p('w'|'k'): 0.004113781166909146
2200 p('x'|'k'): 0.0
2201 p('y'|'k'): 0.005016806301108714
2202 p('z'|'k'): 0.0
2203
2204
2205 Symbol: l
2206 Marginal Probability p('l'): 0.03037010450969173
2207 Transition Probabilities:
2208 p('='|'l'): 0.0
2209 p(' '|'l'): 0.11103216233725523
2210 p('-'|'l'): 0.0011702994712754174
2211 p(', '|'l'): 0.014566048776410106
2212 p('; '|'l'): 0.000355269482351466
2213 p(':', '|'l'): 0.00021943115086414078
2214 p('!'|'l'): 0.0014942216463605777
2215 p('? '|'l'): 0.0007627844768134418
2216 p('/ '|'l'): 0.0
2217 p('. '|'l'): 0.006300808760527471
2218 p(''|'l'): 0.0
2219 p('"'|'l'): 0.0
2220 p('('|'l'): 0.0
2221 p(')'|'l'): 0.00013583833148732524
2222 p('['|'l'): 0.0
2223 p(']'|'l'): 0.0
2224 p('*'|'l'): 0.0
2225 p('0'|'l'): 0.0
2226 p('1'|'l'): 0.0
2227 p('2'|'l'): 0.0
2228 p('3'|'l'): 0.0
2229 p('4'|'l'): 0.0
2230 p('5'|'l'): 0.0
2231 p('6'|'l'): 0.0
2232 p('7'|'l'): 0.0
2233 p('8'|'l'): 0.0

```



```

2234 p('9'|'l'): 0.0
2235 p('a'|'l'): 0.07928778917890954
2236 p('b'|'l'): 0.0007418862719692378
2237 p('c'|'l'): 0.0007314371695471359
2238 p('d'|'l'): 0.07223464504399073
2239 p('e'|'l'): 0.17507471108231804
2240 p('f'|'l'): 0.024074731980522873
2241 p('g'|'l'): 0.000898622808300767
2242 p('h'|'l'): 4.179640968840777e-05
2243 p('i'|'l'): 0.09829470648471296
2244 p('j'|'l'): 0.0
2245 p('k'|'l'): 0.01082527010929761
2246 p('l'|'l'): 0.13803264299596665
2247 p('m'|'l'): 0.004096048149463961
2248 p('n'|'l'): 0.0012747904954964369
2249 p('o'|'l'): 0.0866439572840693
2250 p('p'|'l'): 0.003970658920398737
2251 p('q'|'l'): 1.0449102422101942e-05
2252 p('r'|'l'): 0.0048379344214331986
2253 p('s'|'l'): 0.016237905163946416
2254 p('t'|'l'): 0.017157426177091387
2255 p('u'|'l'): 0.015412426072600363
2256 p('v'|'l'): 0.004419970324549122
2257 p('w'|'l'): 0.004879730831121607
2258 p('x'|'l'): 0.0
2259 p('y'|'l'): 0.10452237152828572
2260 p('z'|'l'): 0.00026122756055254855
2261
2262
2263 Symbol: m
2264 Marginal Probability p('m'): 0.019334911784147646
2265 Transition Probabilities:
2266 p('=|'m'): 0.0
2267 p(' '|'m'): 0.14845391281512604
2268 p('-|'m'): 0.00011488970588235294
2269 p(',|'m'): 0.023585215336134453
2270 p(';|'m'): 0.0010340073529411765
2271 p(':',|'m'): 0.0007385766806722689
2272 p('!|'m'): 0.001953125
2273 p('?|'m'): 0.0016905199579831933
2274 p('/|'m'): 0.0
2275 p('.|'m'): 0.02696625525210084
2276 p(''|'m'): 1.641281512605042e-05
2277 p('"'|'m'): 0.0
2278 p('('|'m'): 0.0
2279 p(')|'m'): 0.0004267331932773109
2280 p('['|'m'): 0.0

```

```

2281 p(']'|'m'): 0.0
2282 p('*'|'m'): 0.0
2283 p('0'|'m'): 0.0
2284 p('1'|'m'): 0.0
2285 p('2'|'m'): 0.0
2286 p('3'|'m'): 0.0
2287 p('4'|'m'): 0.0
2288 p('5'|'m'): 0.0
2289 p('6'|'m'): 0.0
2290 p('7'|'m'): 0.0
2291 p('8'|'m'): 0.0
2292 p('9'|'m'): 0.0
2293 p('a'|'m'): 0.15183495273109243
2294 p('b'|'m'): 0.017807904411764705
2295 p('c'|'m'): 0.0005908613445378151
2296 p('d'|'m'): 0.0
2297 p('e'|'m'): 0.2477842699579832
2298 p('f'|'m'): 0.002182904411764706
2299 p('g'|'m'): 1.641281512605042e-05
2300 p('h'|'m'): 0.0
2301 p('i'|'m'): 0.07910976890756302
2302 p('j'|'m'): 0.0
2303 p('k'|'m'): 9.847689075630252e-05
2304 p('l'|'m'): 0.002658876050420168
2305 p('m'|'m'): 0.02269892331932773
2306 p('n'|'m'): 0.0037257090336134456
2307 p('o'|'m'): 0.10903033088235294
2308 p('p'|'m'): 0.05627954306722689
2309 p('q'|'m'): 0.0
2310 p('r'|'m'): 0.001329438025210084
2311 p('s'|'m'): 0.031496192226890755
2312 p('t'|'m'): 0.0009683560924369748
2313 p('u'|'m'): 0.026999080882352942
2314 p('v'|'m'): 0.0
2315 p('w'|'m'): 9.847689075630252e-05
2316 p('x'|'m'): 0.0
2317 p('y'|'m'): 0.040309873949579834
2318 p('z'|'m'): 0.0
2319
2320
2321 Symbol: n
2322 Marginal Probability p('n'): 0.0576271003566588
2323 Transition Probabilities:
2324 p('= '| 'n'): 0.0
2325 p(' '| 'n'): 0.1924292652840953
2326 p('- '| 'n'): 0.0008535524301463705
2327 p(', '| 'n'): 0.016602971463814883

```

2328 p(';'|'n'): 0.000490104298600174
 2329 p(':',|'n'): 0.00037996850116193265
 2330 p('!'|'n'): 0.0014317653666971375
 2331 p('?'|'n'): 0.0010848376047666772
 2332 p('/'|'n'): 0.0
 2333 p('.'|'n'): 0.01105212727292752
 2334 p(''|'n'): 1.6520369615736202e-05
 2335 p('"'|'n'): 5.506789871912068e-06
 2336 p('('|'n'): 0.0
 2337 p(')'|'n'): 0.00021476480500457065
 2338 p('['|'n'): 0.0
 2339 p(']'|'n'): 0.0
 2340 p('*'|'n'): 0.0
 2341 p('0'|'n'): 0.0
 2342 p('1'|'n'): 0.0
 2343 p('2'|'n'): 0.0
 2344 p('3'|'n'): 0.0
 2345 p('4'|'n'): 0.0
 2346 p('5'|'n'): 0.0
 2347 p('6'|'n'): 0.0
 2348 p('7'|'n'): 0.0
 2349 p('8'|'n'): 0.0
 2350 p('9'|'n'): 0.0
 2351 p('a'|'n'): 0.030298357875260196
 2352 p('b'|'n'): 0.0008535524301463705
 2353 p('c'|'n'): 0.05012280141414364
 2354 p('d'|'n'): 0.19035871229225634
 2355 p('e'|'n'): 0.08161062590173684
 2356 p('f'|'n'): 0.005022192363183806
 2357 p('g'|'n'): 0.13764771963831404
 2358 p('h'|'n'): 0.0007874709516834257
 2359 p('i'|'n'): 0.03128958005220437
 2360 p('j'|'n'): 0.0006608147846294481
 2361 p('k'|'n'): 0.008210623699020893
 2362 p('l'|'n'): 0.011580779100631078
 2363 p('m'|'n'): 0.0007599370023238653
 2364 p('n'|'n'): 0.01073273346035662
 2365 p('o'|'n'): 0.06920382832031895
 2366 p('p'|'n'): 0.0005286518277035585
 2367 p('q'|'n'): 0.0013381499388746324
 2368 p('r'|'n'): 0.000407502450521493
 2369 p('s'|'n'): 0.033998920669185105
 2370 p('t'|'n'): 0.08808110400123352
 2371 p('u'|'n'): 0.005247970747932201
 2372 p('v'|'n'): 0.004146612773549787
 2373 p('w'|'n'): 0.0005837197264226792
 2374 p('x'|'n'): 0.00045155676949678957

```

2375 p('y'|'n'): 0.011393548244986068
2376 p('z'|'n'): 0.00012114937718206549
2377
2378
2379 Symbol: o
2380 Marginal Probability p('o'): 0.060287364364775095
2381 Transition Probabilities:
2382 p('= '| 'o'): 0.0
2383 p(' '| 'o'): 0.13310558646573006
2384 p('- '| 'o'): 0.00041583981218779113
2385 p(', '| 'o'): 0.004453170646973054
2386 p('; '| 'o'): 7.369313127378577e-05
2387 p(': '| 'o'): 7.369313127378577e-05
2388 p('! '| 'o'): 0.0007474589029198271
2389 p('? '| 'o'): 0.0006421830011001332
2390 p('/ '| 'o'): 0.0
2391 p('. '| 'o'): 0.001868647257299568
2392 p('' '| 'o'): 0.0
2393 p('"' '| 'o'): 0.0
2394 p('(' '| 'o'): 0.0
2395 p(')' '| 'o'): 4.7374155818862286e-05
2396 p('[' '| 'o'): 0.0
2397 p(']' '| 'o'): 0.0
2398 p('* '| 'o'): 0.0
2399 p('0 '| 'o'): 0.0
2400 p('1 '| 'o'): 0.0
2401 p('2 '| 'o'): 0.0
2402 p('3 '| 'o'): 0.0
2403 p('4 '| 'o'): 0.0
2404 p('5 '| 'o'): 0.0
2405 p('6 '| 'o'): 0.0
2406 p('7 '| 'o'): 0.0
2407 p('8 '| 'o'): 0.0
2408 p('9 '| 'o'): 0.0
2409 p('a '| 'o'): 0.007258773430467899
2410 p('b '| 'o'): 0.004684777630976382
2411 p('c '| 'o'): 0.00724824584028593
2412 p('d '| 'o'): 0.016059838822594315
2413 p('e '| 'o'): 0.002631897545492349
2414 p('f '| 'o'): 0.08906867673455207
2415 p('g '| 'o'): 0.0040531222200582175
2416 p('h '| 'o'): 0.0018949662327544913
2417 p('i '| 'o'): 0.011754054438168831
2418 p('j '| 'o'): 0.0006421830011001332
2419 p('k '| 'o'): 0.017754780841891388
2420 p('l '| 'o'): 0.03875205945982935
2421 p('m '| 'o'): 0.058207046116108795

```

```

2422 p('n'|'o'): 0.1409381135611153
2423 p('o'|'o'): 0.03788879706490786
2424 p('p'|'o'): 0.01610194918332219
2425 p('q'|'o'): 0.00013159487727461747
2426 p('r'|'o'): 0.10392837027640188
2427 p('s'|'o'): 0.035509561683782775
2428 p('t'|'o'): 0.05277480958221258
2429 p('u'|'o'): 0.12394658300741669
2430 p('v'|'o'): 0.03018260105170626
2431 p('w'|'o'): 0.05216420935165836
2432 p('x'|'o'): 0.0007421951078288424
2433 p('y'|'o'): 0.003658337588234365
2434 p('z'|'o'): 0.0005948088452812709
2435
2436
2437 Symbol: p
2438 Marginal Probability p('p'): 0.014238426042724798
2439 Transition Probabilities:
2440 p('= '| 'p'): 0.0
2441 p(' '| 'p'): 0.05402514041187483
2442 p('- '| 'p'): 0.00015601319425871446
2443 p(', '| 'p'): 0.00849157528751003
2444 p('; '| 'p'): 0.0001783007934385308
2445 p(': '| 'p'): 0.00011143799589908175
2446 p('! '| 'p'): 0.0008469287688330213
2447 p('? '| 'p'): 0.0003566015868770616
2448 p('/ '| 'p'): 0.0
2449 p('. '| 'p'): 0.006039939377730231
2450 p('' '| 'p'): 0.0
2451 p('"' '| 'p'): 0.0
2452 p('(' '| 'p'): 0.0
2453 p(')' '| 'p'): 8.91503967192654e-05
2454 p('[ '| 'p'): 0.0
2455 p('] '| 'p'): 0.0
2456 p('* '| 'p'): 0.0
2457 p('0 '| 'p'): 0.0
2458 p('1 '| 'p'): 0.0
2459 p('2 '| 'p'): 0.0
2460 p('3 '| 'p'): 0.0
2461 p('4 '| 'p'): 0.0
2462 p('5 '| 'p'): 0.0
2463 p('6 '| 'p'): 0.0
2464 p('7 '| 'p'): 0.0
2465 p('8 '| 'p'): 0.0
2466 p('9 '| 'p'): 0.0
2467 p('a '| 'p'): 0.09826602478381029
2468 p('b '| 'p'): 0.00024516359097797983

```

```

2469 p('c'|'p'): 0.0028305250958366767
2470 p('d'|'p'): 4.45751983596327e-05
2471 p('e'|'p'): 0.18249086208433626
2472 p('f'|'p'): 0.002228759917981635
2473 p('g'|'p'): 4.45751983596327e-05
2474 p('h'|'p'): 0.009249353659623785
2475 p('i'|'p'): 0.09365249175358831
2476 p('j'|'p'): 0.0
2477 p('k'|'p'): 0.0004011767852366943
2478 p('l'|'p'): 0.09405366853882499
2479 p('m'|'p'): 0.0006017651778550414
2480 p('n'|'p'): 0.0001783007934385308
2481 p('o'|'p'): 0.10568779531068913
2482 p('p'|'p'): 0.05850494784701792
2483 p('q'|'p'): 0.0
2484 p('r'|'p'): 0.17524739235089595
2485 p('s'|'p'): 0.023914593919942943
2486 p('t'|'p'): 0.04831951502184185
2487 p('u'|'p'): 0.025809039850227332
2488 p('v'|'p'): 0.0
2489 p('w'|'p'): 0.00042346438441651064
2490 p('x'|'p'): 0.0
2491 p('y'|'p'): 0.00751092092359811
2492 p('z'|'p'): 0.0
2493
2494
2495 Symbol: q
2496 Marginal Probability p('q'): 0.0007397203152712736
2497 Transition Probabilities:
2498 p('='|'q'): 0.0
2499 p(' '|'q'): 0.000429000429000429
2500 p('-'|'q'): 0.0
2501 p(', '|'q'): 0.000429000429000429
2502 p('; '|'q'): 0.0
2503 p(':', '|'q'): 0.0
2504 p('!'|'q'): 0.0
2505 p('? '|'q'): 0.0
2506 p('/ '|'q'): 0.0
2507 p('.')|'q'): 0.0
2508 p(''|'q'): 0.0
2509 p('"'|'q'): 0.0
2510 p('('|'q'): 0.0
2511 p(')'|'q'): 0.0
2512 p('['|'q'): 0.0
2513 p(']'|'q'): 0.0
2514 p('*'|'q'): 0.0
2515 p('0'|'q'): 0.0

```

```

2516 p('1'|'q'): 0.0
2517 p('2'|'q'): 0.0
2518 p('3'|'q'): 0.0
2519 p('4'|'q'): 0.0
2520 p('5'|'q'): 0.0
2521 p('6'|'q'): 0.0
2522 p('7'|'q'): 0.0
2523 p('8'|'q'): 0.0
2524 p('9'|'q'): 0.0
2525 p('a'|'q'): 0.0
2526 p('b'|'q'): 0.0
2527 p('c'|'q'): 0.0
2528 p('d'|'q'): 0.0
2529 p('e'|'q'): 0.0
2530 p('f'|'q'): 0.0
2531 p('g'|'q'): 0.0
2532 p('h'|'q'): 0.0
2533 p('i'|'q'): 0.0
2534 p('j'|'q'): 0.0
2535 p('k'|'q'): 0.0
2536 p('l'|'q'): 0.0
2537 p('m'|'q'): 0.0
2538 p('n'|'q'): 0.0
2539 p('o'|'q'): 0.0
2540 p('p'|'q'): 0.0
2541 p('q'|'q'): 0.0
2542 p('r'|'q'): 0.0
2543 p('s'|'q'): 0.0
2544 p('t'|'q'): 0.0
2545 p('u'|'q'): 0.9991419991419992
2546 p('v'|'q'): 0.0
2547 p('w'|'q'): 0.0
2548 p('x'|'q'): 0.0
2549 p('y'|'q'): 0.0
2550 p('z'|'q'): 0.0
2551
2552
2553 Symbol: r
2554 Marginal Probability p('r'): 0.04653573839224598
2555 Transition Probabilities:
2556 p('= '| 'r'): 0.0
2557 p(' '| 'r'): 0.17689899961130093
2558 p('- '| 'r'): 0.0012138322320192577
2559 p(', '| 'r'): 0.019230375810642172
2560 p('; '| 'r'): 0.0006546510914261165
2561 p(':', '| 'r'): 0.0004159762143436782
2562 p('! '| 'r'): 0.0017730133726123988

```

2563 p('?'|'r'): 0.0017730133726123988
 2564 p('/'|'r'): 0.0
 2565 p('.'|'r'): 0.01427275764952981
 2566 p(''|'r'): 0.0
 2567 p('"'|'r'): 0.0
 2568 p('('|'r'): 0.0
 2569 p(')'|'r'): 0.0002386748770824383
 2570 p('['|'r'): 0.0
 2571 p(']'|'r'): 0.0
 2572 p('*'|'r'): 0.0
 2573 p('0'|'r'): 0.0
 2574 p('1'|'r'): 0.0
 2575 p('2'|'r'): 0.0
 2576 p('3'|'r'): 0.0
 2577 p('4'|'r'): 0.0
 2578 p('5'|'r'): 0.0
 2579 p('6'|'r'): 0.0
 2580 p('7'|'r'): 0.0
 2581 p('8'|'r'): 0.0
 2582 p('9'|'r'): 0.0
 2583 p('a'|'r'): 0.05483384818913961
 2584 p('b'|'r'): 0.0014456878268993406
 2585 p('c'|'r'): 0.010010706273057697
 2586 p('d'|'r'): 0.02968433542685297
 2587 p('e'|'r'): 0.24225500023867488
 2588 p('f'|'r'): 0.0030550384266552102
 2589 p('g'|'r'): 0.00951971795448811
 2590 p('h'|'r'): 0.0014388685446969853
 2591 p('i'|'r'): 0.0887802349924647
 2592 p('j'|'r'): 1.363856440471076e-05
 2593 p('k'|'r'): 0.006307836037178727
 2594 p('l'|'r'): 0.008599114857170135
 2595 p('m'|'r'): 0.019960039006294197
 2596 p('n'|'r'): 0.017457362438029773
 2597 p('o'|'r'): 0.10025026765682644
 2598 p('p'|'r'): 0.003709689518081327
 2599 p('q'|'r'): 8.865066863061995e-05
 2600 p('r'|'r'): 0.0335917841288026
 2601 p('s'|'r'): 0.055045245937412626
 2602 p('t'|'r'): 0.0313755174130371
 2603 p('u'|'r'): 0.019728183411414114
 2604 p('v'|'r'): 0.004807593952660543
 2605 p('w'|'r'): 0.0018889411700524403
 2606 p('x'|'r'): 0.0
 2607 p('y'|'r'): 0.03912222199491282
 2608 p('z'|'r'): 0.0005591811405931411
 2609


```

2610
2611 Symbol: s
2612 Marginal Probability p('s'): 0.05118985171003598
2613 Transition Probabilities:
2614 p('='|'s'): 0.0
2615 p(' '|'s'): 0.32053388217644396
2616 p('-'|'s'): 0.00030996410615650706
2617 p(', '|'s'): 0.03767303746226187
2618 p('; '|'s'): 0.0010848743715477747
2619 p(':', '|'s'): 0.00087409877936135
2620 p('!'|'s'): 0.003112039625811331
2621 p('? '|'s'): 0.0016490090447526177
2622 p('/ '|'s'): 6.199282123130141e-06
2623 p('. '|'s'): 0.021269736964459517
2624 p('' '|'s'): 6.199282123130141e-06
2625 p('"'|'s'): 0.0
2626 p('( '|'s'): 0.0
2627 p(')'|'s'): 0.00047114544135789074
2628 p('['|'s'): 0.0
2629 p(']'|'s'): 0.0
2630 p('* '|'s'): 0.0
2631 p('0 '|'s'): 0.0
2632 p('1 '|'s'): 0.0
2633 p('2 '|'s'): 0.0
2634 p('3 '|'s'): 0.0
2635 p('4 '|'s'): 0.0
2636 p('5 '|'s'): 0.0
2637 p('6 '|'s'): 0.0
2638 p('7 '|'s'): 0.0
2639 p('8 '|'s'): 0.0
2640 p('9 '|'s'): 0.0
2641 p('a '|'s'): 0.05199957844881563
2642 p('b '|'s'): 0.0026346949023303102
2643 p('c '|'s'): 0.013799602006087696
2644 p('d '|'s'): 0.0003905547737571989
2645 p('e '|'s'): 0.10877260413244147
2646 p('f '|'s'): 0.0018597846369390425
2647 p('g '|'s'): 0.0002727684134177262
2648 p('h '|'s'): 0.06720641749685387
2649 p('i '|'s'): 0.05622128957466725
2650 p('j '|'s'): 1.8597846369390424e-05
2651 p('k '|'s'): 0.013353253693222326
2652 p('l '|'s'): 0.0074453378298793
2653 p('m '|'s'): 0.010451989659597418
2654 p('n '|'s'): 0.0027214848520541323
2655 p('o '|'s'): 0.05298526430639332
2656 p('p '|'s'): 0.0200732755146954

```

```

2657 p('q'|'s'): 0.0010476786788089938
2658 p('r'|'s'): 0.00015498205307825353
2659 p('s'|'s'): 0.06002144951614603
2660 p('t'|'s'): 0.11014884476377636
2661 p('u'|'s'): 0.02446236725787154
2662 p('v'|'s'): 0.0004897432877272812
2663 p('w'|'s'): 0.004147319740374065
2664 p('x'|'s'): 0.0
2665 p('y'|'s'): 0.0023061329498044124
2666 p('z'|'s'): 2.4797128492520565e-05
2667
2668
2669 Symbol: t
2670 Marginal Probability p('t'): 0.07049398148192224
2671 Transition Probabilities:
2672 p('='|'t'): 0.0
2673 p(' '|'t'): 0.2319528225443414
2674 p('-'|'t'): 0.00046817322409291437
2675 p(', '|'t'): 0.014535878274961736
2676 p('; '|'t'): 0.0004951832177905825
2677 p(':', '|'t'): 0.0002746016025929594
2678 p('!'|'t'): 0.0015935896281624202
2679 p('? '|'t'): 0.001971729539929774
2680 p('/ '|'t'): 0.0
2681 p('. '|'t'): 0.011119114072206716
2682 p(''|'t'): 0.0
2683 p('"'|'t'): 0.0
2684 p('('|'t'): 0.0
2685 p(')'|'t'): 0.000198073287116233
2686 p('['|'t'): 0.0
2687 p(']'|'t'): 0.0
2688 p('*'|'t'): 0.0
2689 p('0'|'t'): 0.0
2690 p('1'|'t'): 0.0
2691 p('2'|'t'): 0.0
2692 p('3'|'t'): 0.0
2693 p('4'|'t'): 0.0
2694 p('5'|'t'): 0.0
2695 p('6'|'t'): 0.0
2696 p('7'|'t'): 0.0
2697 p('8'|'t'): 0.0
2698 p('9'|'t'): 0.0
2699 p('a'|'t'): 0.03414063203385252
2700 p('b'|'t'): 6.752498424417034e-05
2701 p('c'|'t'): 0.0028540560007202665
2702 p('d'|'t'): 1.3504996848834069e-05
2703 p('e'|'t'): 0.08616638156117763

```

2704 p('f'|'t'): 0.0008373098046277123
 2705 p('g'|'t'): 4.051499054650221e-05
 2706 p('h'|'t'): 0.33208337084721345
 2707 p('i'|'t'): 0.06256865040064824
 2708 p('j'|'t'): 0.0
 2709 p('k'|'t'): 4.501665616278023e-05
 2710 p('l'|'t'): 0.013500495183217791
 2711 p('m'|'t'): 0.0012334563788601782
 2712 p('n'|'t'): 0.0008012964796974881
 2713 p('o'|'t'): 0.10678401008373098
 2714 p('p'|'t'): 0.0001575582965697308
 2715 p('q'|'t'): 0.0
 2716 p('r'|'t'): 0.025465922391284774
 2717 p('s'|'t'): 0.01755649590348429
 2718 p('t'|'t'): 0.01969928873683263
 2719 p('u'|'t'): 0.01334293688664806
 2720 p('v'|'t'): 7.652831547672639e-05
 2721 p('w'|'t'): 0.00523543711173134
 2722 p('x'|'t'): 0.0
 2723 p('y'|'t'): 0.014103718375799046
 2724 p('z'|'t'): 0.0006167281894300891
 2725
 2726
 2727 Symbol: u
 2728 Marginal Probability p('u'): 0.020363729142409966
 2729 Transition Probabilities:
 2730 p('= '| 'u'): 0.0
 2731 p(' '| 'u'): 0.04866760168302945
 2732 p('- '| 'u'): 4.675081813931744e-05
 2733 p(', '| 'u'): 0.004815334268349696
 2734 p('; '| 'u'): 0.00014025245441795232
 2735 p(': '| 'u'): 7.791803023219573e-05
 2736 p('! '| 'u'): 0.0016362786348761104
 2737 p('? '| 'u'): 0.001496026180458158
 2738 p('/ '| 'u'): 0.0
 2739 p('. '| 'u'): 0.0030543867851020724
 2740 p('' '| 'u'): 0.0
 2741 p('"' '| 'u'): 0.0
 2742 p('('| 'u'): 0.0
 2743 p(')' '| 'u'): 4.675081813931744e-05
 2744 p('['| 'u'): 0.0
 2745 p('] '| 'u'): 0.0
 2746 p('* '| 'u'): 0.0
 2747 p('0 '| 'u'): 0.0
 2748 p('1 '| 'u'): 0.0
 2749 p('2 '| 'u'): 0.0
 2750 p('3 '| 'u'): 0.0

```

2751 p('4'|'u'): 0.0
2752 p('5'|'u'): 0.0
2753 p('6'|'u'): 0.0
2754 p('7'|'u'): 0.0
2755 p('8'|'u'): 0.0
2756 p('9'|'u'): 0.0
2757 p('a'|'u'): 0.02060152719339255
2758 p('b'|'u'): 0.013978494623655914
2759 p('c'|'u'): 0.03193080878915381
2760 p('d'|'u'): 0.022004051737572073
2761 p('e'|'u'): 0.029499766245909304
2762 p('f'|'u'): 0.005142589995324918
2763 p('g'|'u'): 0.05064671965092722
2764 p('h'|'u'): 0.0006545114539504442
2765 p('i'|'u'): 0.027271310581268506
2766 p('j'|'u'): 1.5583606046439145e-05
2767 p('k'|'u'): 0.001496026180458158
2768 p('l'|'u'): 0.09770920991117345
2769 p('m'|'u'): 0.021271622253389435
2770 p('n'|'u'): 0.13662147420913198
2771 p('o'|'u'): 0.002306373694872994
2772 p('p'|'u'): 0.04324450677886863
2773 p('q'|'u'): 1.5583606046439145e-05
2774 p('r'|'u'): 0.12457534673523453
2775 p('s'|'u'): 0.13691756272401434
2776 p('t'|'u'): 0.1707028206326944
2777 p('u'|'u'): 0.0
2778 p('v'|'u'): 0.0012466884837151317
2779 p('w'|'u'): 3.116721209287829e-05
2780 p('x'|'u'): 0.0006700950599968833
2781 p('y'|'u'): 0.0003116721209287829
2782 p('z'|'u'): 0.001153186847436497
2783
2784
2785 Symbol: v
2786 Marginal Probability p('v'): 0.008339703940510111
2787 Transition Probabilities:
2788 p('='|'v'): 0.0
2789 p(' '|'v'): 0.06643835616438357
2790 p('-'|'v'): 0.00041856925418569255
2791 p(', '|'v'): 0.021575342465753426
2792 p('; '|'v'): 0.00041856925418569255
2793 p(':', '|'v'): 0.00022831050228310502
2794 p('!'|'v'): 0.00106544901065449
2795 p('? '|'v'): 0.0009512937595129375
2796 p('/ '|'v'): 0.0
2797 p('. '|'v'): 0.010616438356164383

```

```

2798 p(''|'v'): 0.0
2799 p('"'|'v'): 0.0
2800 p(' '|'v'): 0.0
2801 p(')'|'v'): 0.00022831050228310502
2802 p('['|'v'): 0.0
2803 p(']'|'v'): 0.0
2804 p('*'|'v'): 0.0
2805 p('0'|'v'): 0.0
2806 p('1'|'v'): 0.0
2807 p('2'|'v'): 0.0
2808 p('3'|'v'): 0.0
2809 p('4'|'v'): 0.0
2810 p('5'|'v'): 0.0
2811 p('6'|'v'): 0.0
2812 p('7'|'v'): 0.0
2813 p('8'|'v'): 0.0
2814 p('9'|'v'): 0.0
2815 p('a'|'v'): 0.06430745814307458
2816 p('b'|'v'): 0.0
2817 p('c'|'v'): 0.0
2818 p('d'|'v'): 0.0
2819 p('e'|'v'): 0.5516742770167428
2820 p('f'|'v'): 0.0
2821 p('g'|'v'): 0.00011415525114155251
2822 p('h'|'v'): 0.0
2823 p('i'|'v'): 0.17937595129375952
2824 p('j'|'v'): 0.0
2825 p('k'|'v'): 0.00030441400304414006
2826 p('l'|'v'): 0.008181126331811263
2827 p('m'|'v'): 0.0
2828 p('n'|'v'): 0.020243531202435314
2829 p('o'|'v'): 0.05570776255707763
2830 p('p'|'v'): 0.0
2831 p('q'|'v'): 0.0
2832 p('r'|'v'): 0.003310502283105023
2833 p('s'|'v'): 0.00993150684931507
2834 p('t'|'v'): 7.610350076103501e-05
2835 p('u'|'v'): 0.00076103500761035
2836 p('v'|'v'): 3.805175038051751e-05
2837 p('w'|'v'): 3.805175038051751e-05
2838 p('x'|'v'): 0.0
2839 p('y'|'v'): 0.003995433789954338
2840 p('z'|'v'): 0.0
2841
2842
2843 Symbol: w
2844 Marginal Probability p('w'): 0.01873037845056044

```

```

2845 Transition Probabilities:
2846 p('='|'w'): 0.0
2847 p(' '| 'w'): 0.11048235433644511
2848 p('-'|'w'): 0.0003727360520475069
2849 p(', '| 'w'): 0.01650204157701235
2850 p('; '| 'w'): 0.0004405062433288718
2851 p(':', '| 'w'): 0.00025413821730511834
2852 p('!'| 'w'): 0.0015417718516510513
2853 p('? '| 'w'): 0.001440116564729004
2854 p('/ '| 'w'): 0.0
2855 p('. '| 'w'): 0.009132033275163919
2856 p('' '| 'w'): 0.0
2857 p('"'| 'w'): 0.0
2858 p('('| 'w'): 0.0
2859 p(')'| 'w'): 0.0002880233129458008
2860 p('['| 'w'): 0.0
2861 p(']'| 'w'): 0.0
2862 p('* '| 'w'): 0.0
2863 p('0 '| 'w'): 0.0
2864 p('1 '| 'w'): 0.0
2865 p('2 '| 'w'): 0.0
2866 p('3 '| 'w'): 0.0
2867 p('4 '| 'w'): 0.0
2868 p('5 '| 'w'): 0.0
2869 p('6 '| 'w'): 0.0
2870 p('7 '| 'w'): 0.0
2871 p('8 '| 'w'): 0.0
2872 p('9 '| 'w'): 0.0
2873 p('a '| 'w'): 0.20613997933009165
2874 p('b '| 'w'): 6.777019128136489e-05
2875 p('c '| 'w'): 0.00027108076512545956
2876 p('d '| 'w'): 0.006217915050065229
2877 p('e '| 'w'): 0.13142334344238688
2878 p('f '| 'w'): 0.0007454721040950138
2879 p('g '| 'w'): 0.0
2880 p('h '| 'w'): 0.2020907104010301
2881 p('i '| 'w'): 0.1697812717076394
2882 p('j '| 'w'): 0.0
2883 p('k '| 'w'): 0.001067380512681497
2884 p('l '| 'w'): 0.0033715670162479034
2885 p('m '| 'w'): 3.3885095640682445e-05
2886 p('n '| 'w'): 0.03762939870897786
2887 p('o '| 'w'): 0.0778848923301086
2888 p('p '| 'w'): 1.6942547820341222e-05
2889 p('q '| 'w'): 0.0
2890 p('r '| 'w'): 0.008945665249140165
2891 p('s '| 'w'): 0.012368059908849092

```

```

2892 p('t'|'w'): 0.00011859783474238856
2893 p('u'|'w'): 0.00032190840858648326
2894 p('v'|'w'): 0.0
2895 p('w'|'w'): 0.0003896785998678481
2896 p('x'|'w'): 0.0
2897 p('y'|'w'): 0.0006607593649933077
2898 p('z'|'w'): 0.0
2899
2900
2901 Symbol: x
2902 Marginal Probability p('x'): 0.0013747183207872833
2903 Transition Probabilities:
2904 p('= '| 'x'): 0.0
2905 p(' '| 'x'): 0.07156048014773776
2906 p('- '| 'x'): 0.0018467220683287165
2907 p(', '| 'x'): 0.008541089566020314
2908 p('; '| 'x'): 0.0
2909 p(': '| 'x'): 0.0006925207756232687
2910 p('! '| 'x'): 0.0004616805170821791
2911 p('? '| 'x'): 0.0
2912 p('/ '| 'x'): 0.00023084025854108956
2913 p('. '| 'x'): 0.0043859649122807015
2914 p('' '| 'x'): 0.0
2915 p('"' '| 'x'): 0.0
2916 p('(' '| 'x'): 0.0
2917 p(')' '| 'x'): 0.0
2918 p('[' '| 'x'): 0.0
2919 p(']' '| 'x'): 0.0
2920 p('* '| 'x'): 0.0
2921 p('0 '| 'x'): 0.0
2922 p('1 '| 'x'): 0.0
2923 p('2 '| 'x'): 0.0
2924 p('3 '| 'x'): 0.0
2925 p('4 '| 'x'): 0.0
2926 p('5 '| 'x'): 0.0
2927 p('6 '| 'x'): 0.0
2928 p('7 '| 'x'): 0.0
2929 p('8 '| 'x'): 0.0
2930 p('9 '| 'x'): 0.0
2931 p('a '| 'x'): 0.07779316712834719
2932 p('b '| 'x'): 0.0
2933 p('c '| 'x'): 0.13896583564173592
2934 p('d '| 'x'): 0.0
2935 p('e '| 'x'): 0.06209602954755309
2936 p('f '| 'x'): 0.0
2937 p('g '| 'x'): 0.0
2938 p('h '| 'x'): 0.013619575253924284

```

```

2939 p('i'|'x'): 0.11842105263157894
2940 p('j'|'x'): 0.0
2941 p('k'|'x'): 0.0
2942 p('l'|'x'): 0.0004616805170821791
2943 p('m'|'x'): 0.0
2944 p('n'|'x'): 0.0
2945 p('o'|'x'): 0.0011542012927054479
2946 p('p'|'x'): 0.3127885503231764
2947 p('q'|'x'): 0.0009233610341643582
2948 p('r'|'x'): 0.0
2949 p('s'|'x'): 0.0
2950 p('t'|'x'): 0.10872576177285319
2951 p('u'|'x'): 0.0039242843951985225
2952 p('v'|'x'): 0.036011080332409975
2953 p('w'|'x'): 0.0
2954 p('x'|'x'): 0.03647276084949215
2955 p('y'|'x'): 0.0009233610341643582
2956 p('z'|'x'): 0.0
2957
2958
2959 Symbol: y
2960 Marginal Probability p('y'): 0.01446151629653677
2961 Transition Probabilities:
2962 p('='|'y'): 0.0
2963 p(' '|'y'): 0.5563625990213074
2964 p('-'|'y'): 0.0032915670053323388
2965 p(', '|'y'): 0.06885958175155252
2966 p('; '|'y'): 0.001996883983234952
2967 p(':', '|'y'): 0.00223826556362599
2968 p('! '|'y'): 0.005837045489456014
2969 p('? '|'y'): 0.004783744047749666
2970 p('/ '|'y'): 2.1943780035548923e-05
2971 p('. '|'y'): 0.04733273353667903
2972 p('' '|'y'): 0.0
2973 p('"' '|'y'): 0.0
2974 p('(' '|'y'): 0.0
2975 p(')' '|'y'): 0.0008119198613153102
2976 p('[' '|'y'): 0.0
2977 p(']' '|'y'): 0.0
2978 p('* '|'y'): 0.0
2979 p('0 '|'y'): 0.0
2980 p('1 '|'y'): 0.0
2981 p('2 '|'y'): 0.0
2982 p('3 '|'y'): 0.0
2983 p('4 '|'y'): 0.0
2984 p('5 '|'y'): 0.0
2985 p('6 '|'y'): 0.0

```



```

2986 p('7'|'y'): 0.0
2987 p('8'|'y'): 0.0
2988 p('9'|'y'): 0.0
2989 p('a'|'y'): 0.02563033508152114
2990 p('b'|'y'): 0.0032696232252967896
2991 p('c'|'y'): 0.00366461126593667
2992 p('d'|'y'): 0.00021943780035548925
2993 p('e'|'y'): 0.056154133110969694
2994 p('f'|'y'): 0.0021504904434837945
2995 p('g'|'y'): 0.0001974940203199403
2996 p('h'|'y'): 0.0002633253604265871
2997 p('i'|'y'): 0.02361150731825064
2998 p('j'|'y'): 0.0
2999 p('k'|'y'): 0.0007899760812797612
3000 p('l'|'y'): 0.004432643567180883
3001 p('m'|'y'): 0.0036207237058655723
3002 p('n'|'y'): 0.0009216387614930548
3003 p('o'|'y'): 0.13139935485286697
3004 p('p'|'y'): 0.001141076561848544
3005 p('q'|'y'): 0.0
3006 p('r'|'y'): 0.0019529964231638542
3007 p('s'|'y'): 0.028417195146035856
3008 p('t'|'y'): 0.017445305128261396
3009 p('u'|'y'): 0.001163020341884093
3010 p('v'|'y'): 2.1943780035548923e-05
3011 p('w'|'y'): 0.001777446182879463
3012 p('x'|'y'): 0.0
3013 p('y'|'y'): 0.0
3014 p('z'|'y'): 0.00021943780035548925
3015
3016
3017 Symbol: z
3018 Marginal Probability p('z'): 0.0007092556433424696
3019 Transition Probabilities:
3020 p('= '| 'z'): 0.0
3021 p(' '| 'z'): 0.0348993288590604
3022 p('- '| 'z'): 0.0022371364653243847
3023 p(', '| 'z'): 0.013870246085011185
3024 p('; '| 'z'): 0.00044742729306487697
3025 p(': '| 'z'): 0.00044742729306487697
3026 p('! '| 'z'): 0.0013422818791946308
3027 p('? '| 'z'): 0.0008948545861297539
3028 p('/ '| 'z'): 0.0
3029 p('. '| 'z'): 0.006263982102908278
3030 p('' '| 'z'): 0.0
3031 p('" '| 'z'): 0.0
3032 p('(' '| 'z'): 0.0

```

3033 p(')'|'z'): 0.00044742729306487697
 3034 p('['|'z'): 0.0
 3035 p(']'|'z'): 0.0
 3036 p('*'|'z'): 0.0
 3037 p('0'|'z'): 0.0
 3038 p('1'|'z'): 0.0
 3039 p('2'|'z'): 0.0
 3040 p('3'|'z'): 0.0
 3041 p('4'|'z'): 0.0
 3042 p('5'|'z'): 0.0
 3043 p('6'|'z'): 0.0
 3044 p('7'|'z'): 0.0
 3045 p('8'|'z'): 0.0
 3046 p('9'|'z'): 0.0
 3047 p('a'|'z'): 0.04429530201342282
 3048 p('b'|'z'): 0.0
 3049 p('c'|'z'): 0.0
 3050 p('d'|'z'): 0.012080536912751677
 3051 p('e'|'z'): 0.3261744966442953
 3052 p('f'|'z'): 0.0
 3053 p('g'|'z'): 0.0
 3054 p('h'|'z'): 0.06890380313199106
 3055 p('i'|'z'): 0.10782997762863535
 3056 p('j'|'z'): 0.0
 3057 p('k'|'z'): 0.0017897091722595079
 3058 p('l'|'z'): 0.019686800894854587
 3059 p('m'|'z'): 0.028187919463087248
 3060 p('n'|'z'): 0.007606263982102908
 3061 p('o'|'z'): 0.28232662192393737
 3062 p('p'|'z'): 0.0
 3063 p('q'|'z'): 0.0
 3064 p('r'|'z'): 0.0
 3065 p('s'|'z'): 0.0008948545861297539
 3066 p('t'|'z'): 0.0
 3067 p('u'|'z'): 0.011633109619686801
 3068 p('v'|'z'): 0.0
 3069 p('w'|'z'): 0.00044742729306487697
 3070 p('x'|'z'): 0.0
 3071 p('y'|'z'): 0.006263982102908278
 3072 p('z'|'z'): 0.021029082774049218