Probabilistic and Unsupervised Learning Coursework

November 22, 2021

1 Non-bonus Questions

1. Models for binary vectors

(a) Explain why a multivariate Gaussian would not be an appropriate model for this data set of images.

- Because this is a binary data set, we cannot use multivariate Gaussian models on discrete r.v.. The Gaussian model is used for continuous r.v.. If we have colored pictures, we might try multivariate Gaussian models.

(b) What is the equation for the maximum likelihood (ML) estimate of $\boldsymbol{p}$? Note that you can solve for $\boldsymbol{p}$ directly.

$$\mathbb{P}(\underline{x}|\underline{p}) = \prod_{d=1}^{D} p_d^{x_d}(1-p_d)^{1-x_d} \tag{1}$$

$$\begin{aligned}
\mathbb{L}(\underline{p};\underline{x}) &= f_{\underline{x}}(\underline{x};\underline{p}) && (2)\\
&= \prod_{n=1}^{N} \mathbb{P}(X_n|\underline{p}) && (3)\\
&= \prod_{n=1}^{N}\prod_{d=1}^{D} p_d^{x_d^{(n)}}(1-p_d)^{1-x_d^{(n)}} && (4)
\end{aligned}$$

$$l(\underline{p};\underline{x}) = log(\mathbb{L}(\underline{p};\underline{x})) \tag{5}$$

$$= \sum_{n=1}^{N} log(\prod_{d=1}^{D} p_d^{x_d^{(n)}}(1-p_d)^{1-x_d^{(n)}}) \tag{6}$$

$$= \sum_{n=1}^{N}\sum_{d=1}^{D} x_d^{(n)} log(p_d) + (1-x_d^{(n)}) log(1-p_d) \tag{7}$$

Let $\frac{\partial l}{\partial \underline{p}}$ be zero, then find $\hat{p}$.

$$\frac{\partial l(\underline{p};\underline{x})}{\partial p_d} = \sum_{n=1}^{N}\sum_{d=1}^{D} \frac{x_d^{(n)}}{p_d} - \frac{x_d^{(n)}-1}{p_d-1} \tag{8}$$

$$= \sum_{d=1}^{D} \frac{\sum_{n=1}^{N} x_d^{(n)}}{p_d} - \frac{\sum_{n=1}^{N} x_d^{(n)} - N}{p_d - 1} \tag{9}$$

$$\implies \sum_{d=1}^{D} \frac{\sum_{n=1}^{N} x_d^{(n)}}{p_d} - \frac{\sum_{n=1}^{N} x_d^{(n)} - N}{p_d - 1} = 0$$

$$\implies \frac{\sum_{n=1}^{N} x_d^{(n)}}{p_d} - \frac{\sum_{n=1}^{N} x_d^{(n)} - N}{p_d - 1} = 0$$

$$\implies (\hat{p_d} - 1)\sum_{n=1}^{N} x_d^{(n)} - \hat{p_d}(\sum_{n=1}^{N} x_d^{(n)} - N) = 0$$

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

(c) Find the maximum a posteriori (MAP) estimator for $\boldsymbol{p}$.

$$\pi(\underline{x}|\underline{p}) \propto \prod_{n=1}^{N} \mathbb{P}(\underline{X_n}|\underline{p})\mathbb{P}(\underline{p}) \tag{10}$$

$$\propto \prod_{n=1}^{N}\prod_{d=1}^{D} p_d^{x_d^{(n)}}(1-p_d)^{1-x_d^{(n)}} \prod_{d=1}^{D} \frac{1}{B(\alpha,\beta)} p_d^{\alpha-1}(1-p_d)^{\beta-1} \tag{11}$$

$$log(\pi(\underline{x}|\underline{p})) = \sum_{n=1}^{N}\sum_{d=1}^{D}(x_d^{(n)} log(p_d) + (1-x_d^{(n)}) log(1-p_d)) + \sum_{d=1}^{D} log(\frac{p_d^{\alpha-1}(1-p_d)^{\beta-1}}{B(\alpha,\beta)})$$

$$= \sum_{n=1}^{N}\sum_{d=1}^{D}(x_d^{(n)} log(p_d) + (1-x_d^{(n)}) log(1-p_d))$$

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

Let $\frac{\partial log(\pi(\underline{x}|\underline{p}))}{\partial \underline{p}}$ be zero, then find $\hat{p}$.

$$\frac{\partial log(\pi(\underline{x}|\underline{p}))}{\partial p_d} = \sum_{n=1}^{N}\sum_{d=1}^{D}\frac{x_d^{(n)}}{p_d} - \frac{x_d^{(n)}-1}{p_d-1} + (\alpha-1)\sum_{d=1}^{D}\frac{1}{p_d} + (\beta-1)\sum_{d=1}^{D}\frac{1}{p_d-1}$$

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

$$= \sum_{d=1}^{D}((\sum_{n=1}^{N} x_d^{(n)} + \alpha - 1)(1-p_d) + (\sum_{n=1}^{N} x_d^{(n)} - N - \beta + 1)p_d)$$

$$\implies (\sum_{n=1}^{N} x_d^{(n)} + \alpha - 1)(1-\hat{p}) = \hat{p}(-\sum_{n=1}^{N} x_d^{(n)} + N + \beta - 1)$$

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

(d) Write code to learn the ML parameters of a multivariate Bernoulli from this data set and display these parameters as an 8×8 image.

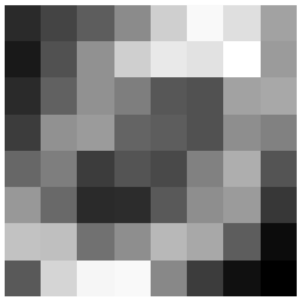


Figure 1: Image for $\hat{p}^{ML} = \frac{\sum_{n=1}^{N} x_d^{(n)}}{N}$

(e) Modify your code to learn MAP parameters with $\alpha = \beta = 3$. Show the new learned parameter vector for this data set as an image.

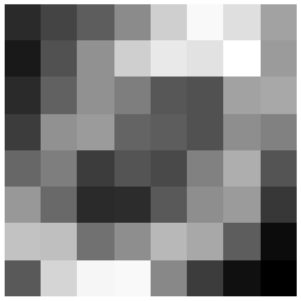


Figure 2: Image for $p^{MAP} = \frac{\sum_{n=1}^{N} x_d^{(n)}+\alpha-1}{\alpha+\beta+N-2}$

- Explain why this might be better or worse than the ML estimate.
  A: ML only uses the probability of observations and find the best estimates based on the likelihood.
  MAP considers the prior information and applies Bayes' formula to find the best estimates.
  Hence, I may conclude that if we have some prior information, we can try MAP. If we don't have, we use ML.

2. Model selection.

(a) all D components are generated from a Bernoulli distribution with $p_d = 0.5$

(b) all D components are generated from Bernoulli distributions with unknown, but identical, p_d

(c) each component is Bernoulli distributed with separate, unknown p_d

Assume that all three models are equally likely a priori, and take the prior distributions for any unknown probablities to be uniform.

Model A: $p_d = \frac{1}{2}$

Model B: $p^b \sim U(0,1)$

Model C: $p_d^c \sim U(0,1)$

$$P(\underline{x}|M_a) = \prod_{n=1}^{N}\prod_{d=1}^{D}\left(\frac{1}{2}\right)^{x_d^{(n)}}\left(1-\frac{1}{2}\right)^{1-x_d^{(n)}}$$
$$= \prod_{n=1}^{N}\prod_{d=1}^{D}\left(\frac{1}{2}\right)^{1}$$
$$= \left(\frac{1}{2}\right)^{ND}$$

$$P(\underline{x}|M_b) = \int_0^1 P(\underline{x}|p^b, M_b)P(p^b|M_b)\,dp^b$$
$$= \int_0^1 \prod_{n=1}^{N}\left(\prod_{d=1}^{D} p^{x_d^{(n)}}(1-p)^{1-x_d^{(n)}}\right)\cdot 1\,dp$$
$$= \int_0^1 p^{\sum_{n=1}^{N}\sum_{d=1}^{D} x_d^{(n)}}(1-p)^{\sum_{n=1}^{N}\sum_{d=1}^{D}(1-x_d^{(n)})}\,dp$$

Beta function $B(\alpha,\beta) = \int_0^1 \pi^{\alpha-1}(1-\pi)^{\beta-1}\,d\pi$

$$= B\left(\left(\sum_{n=1}^{N}\sum_{d=1}^{D} x_d^{(n)}\right)+1, \left(\sum_{n=1}^{N}\sum_{d=1}^{D}(1-x_d^{(n)})\right)+1\right)$$

$$P(\underline{x}|M_c) = \int_0^1\int_0^1\cdots\int_0^1 P(\underline{x}|\underline{p}, M_c)\cdot P(\underline{p}^c|M_c)\,d\underline{p}^c$$
$$= \prod_{d=1}^{D}\int_0^1 \prod_{n=1}^{N} p_d^{x_d^{(n)}}(1-p_d)^{1-x_d^{(n)}}\,dp_d$$
$$= \prod_{d=1}^{D}\int_0^1 p_d^{\sum_{n=1}^{N} x_d^{(n)}}(1-p_d)^{\sum_{n=1}^{N}(1-x_d^{(n)})}\,dp_d$$

Beta function

$$= \prod_{d=1}^{D} B\left(\sum_{n=1}^{N} x_d^{(n)}+1,\ \sum_{n=1}^{N}(1-x_d^{(n)})+1\right)$$

$P(M_A) = P(M_b) = P(M_c) = \frac{1}{3}$ $\quad P(M_k|\underline{x}) = \frac{P(\underline{x}|M_k)P(M_k)}{\underbrace{P(\underline{x})}_{\text{constant}}} \propto P(\underline{x}|M_k)\cdot P(M_k)$

Since the posterior probabilities for them are very small, we calculate log instead.

$\log(P(M_a|\underline{x})) \propto \log(P(M_a|\underline{x})) + \log(P(M_a)) = ND\log(\frac{1}{2}) + \log(\frac{1}{3})$

$\log(P(M_b|\underline{x})) \propto \log(P(M_b|\underline{x})) + \log(P(M_b)) = \text{Betaln}\left(\left(\sum_{n=1}^{N}\sum_{d=1}^{D} x_d^{(n)}\right)+1, \left(\sum_{n=1}^{N}\sum_{d=1}^{D}(1-x_d^{(n)})\right)+1\right) + \log(\frac{1}{3})$

$\log(P(M_c|\underline{x})) \propto \log(P(M_c|\underline{x})) + \log(P(M_c)) = \sum_{d=1}^{D}\text{Betaln}\left(\left(\sum_{n=1}^{N} x_d^{(n)}\right)+1, \sum_{n=1}^{N}(1-x_d^{(n)})+1\right) + \log(\frac{1}{3})$

Calculate the posterior probabilities of each of the three models having generated the data in binarydigits.txt.

- The log posterior probabilities for model A is -4437.240567872318
- The log posterior probabilities for model B is -4284.819954866027
- The log posterior probabilities for model C is -3852.2943562097994

3. EM for Binary Data.

(abc)

3. (a) $P(x^{(n)} \mid s^{(n)}=k, \pi, p) = \prod_{d=1}^{D} p_{kd}^{x_d^{(n)}} (1-p_{kd})^{1-x_d^{(n)}}$

$$P(X \mid \pi, p) = \prod_{n=1}^{N} P(x^{(n)} \mid \pi, p)$$
$$= \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \cdot \prod_{d=1}^{D} p_{kd}^{x_d^{(n)}} (1-p_{kd})^{1-x_d^{(n)}}$$

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

$$r_{nk} = P(s^{(n)}=k \mid x^{(n)}, \pi, p)$$
$$= \frac{P(s^{(n)}=k \mid \pi, p) \cdot P(x^{(n)} \mid s^{(n)}=k, \pi, p)}{\sum_i P(x^{(n)} \mid s^{(n)}=i, \pi, p) \cdot P(s^{(n)}=i \mid \pi, p)}$$
$$= \frac{\pi_k \cdot \prod_{d=1}^{D} p_{kd}^{x_d^{(n)}} (1-p_{kd})^{1-x_d^{(n)}}}{\sum_{i=1}^{K} \pi_i \cdot \prod_{d=1}^{D} p_{id}^{x_d^{(n)}} (1-p_{id})^{1-x_d^{(n)}}}$$

(c) E step: $\langle \sum_n \log P(x^{(n)}, s^{(n)} \mid \pi, p) \rangle_{q(\{s^{(n)}\})}$

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

$$\left. \frac{\partial \langle \dots \rangle_q}{\partial p_{kd}} \right|_{\hat{p}_{kd}} = \sum_{n=1}^{N} r_{nk} \left\{ \frac{x_d^{(n)}}{\hat{p}_{kd}} - \frac{1-x_d^{(n)}}{1-\hat{p}_{kd}} \right\} = 0.$$

$$\Rightarrow \sum_n r_{nk} \cdot \frac{x_d^{(n)}}{\hat{p}_{kd}} = \sum_n \frac{1-x_d^{(n)}}{1-\hat{p}_{kd}} \cdot r_{nk}.$$

$$\Rightarrow (1-\hat{p}_{kd}) \sum_n r_{nk} \cdot x_d^{(n)} = \hat{p}_{kd} \sum_n r_{nk} (1-x_d^{(n)})$$

$$\Rightarrow \sum_n r_{nk} \cdot x_d^{(n)} = \hat{p}_{kd} \sum_n r_{nk}$$

$$\Rightarrow \hat{p}_{kd} = \frac{\sum_n r_{nk} \cdot x_d^{(n)}}{\sum_n r_{nk}}$$

$$\langle \sum_n \log P(x^{(n)}, s^{(n)} | \Pi, p \rangle_{q(\{s^{(n)}\})} = \sum_k \sum_n r_{nk} \log(\pi_k) + \sum_{d=1}^{D} \left(x_d^{(n)} \cdot \log(p_{kd}) + (1 - x_d^{(n)}) \log(1 - p_{kd}) \right)$$

$\sum_k \pi_k = 1$ Use Lagrange multiplier λ. we have

$$\left. \frac{\partial [\langle \cdots \rangle_q + \lambda(\sum_k \pi_k - 1)]}{\partial \pi_k} \right|_{\hat{\pi}_k} = 0 \Rightarrow \sum_n r_{nk} \frac{1}{\hat{\pi}_k} + \lambda = 0$$

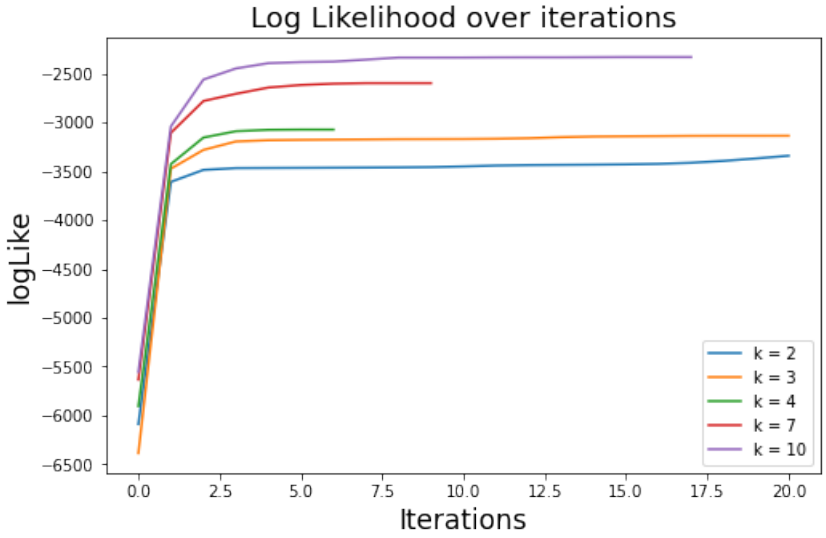
$$\Rightarrow \underbrace{\sum_k \sum_n r_{nk}}_{=N} + \sum_k \lambda \underbrace{\hat{\pi}_k}_{=1} = 0.$$

$$\Rightarrow N + \lambda = 0$$

$$\Rightarrow \lambda = -N$$

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

(d) Implement the EM algorithm for a mixture of K multivariate Bernoullis. Run your algorithm on the data set for values of K in $\{2, 3, 4, 7, 10\}$. Plot the log likelihood as a function of the iteration number, and display the parameters found.



- The parameters π_k for $k = 2$ are [0.10257364 0.89742636]
- The parameters π_k for $k = 3$ are [0.45531926 0.53650443 0.00817631]
- The parameters π_k for $k = 4$ are [0.47080221 0.09799633 0.30995424 0.12124722]
- The parameters π_k for $k = 7$ are [0.13957608 0.05205032 0.23497911 0.2568323 0.06252454 0.00183024 0.25220741]

- The parameters π_k for $k = 10$ are [0.04788091 0.00483567 0.05966123 0.11369644 0.14214807 0.05795266 0.15126459 0.11786337 0.15949591 0.14520116]

(e) Run the algorithm a few times starting from randomly chosen initial conditions.

- Do you obtain the same solutions (up to permutation)?
  A: No, I got different solutions each time. Both the parameter and the log likelihood are different.
- Does this depend on K? Show the learned probability vectors as images.
  A: It depends on K. I visualize P_k for each k and I got the plots below.

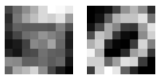


Figure 3: $P_{k=2}$

Figure 4: $P_{k=3}$

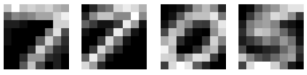


Figure 5: $P_{k=4}$

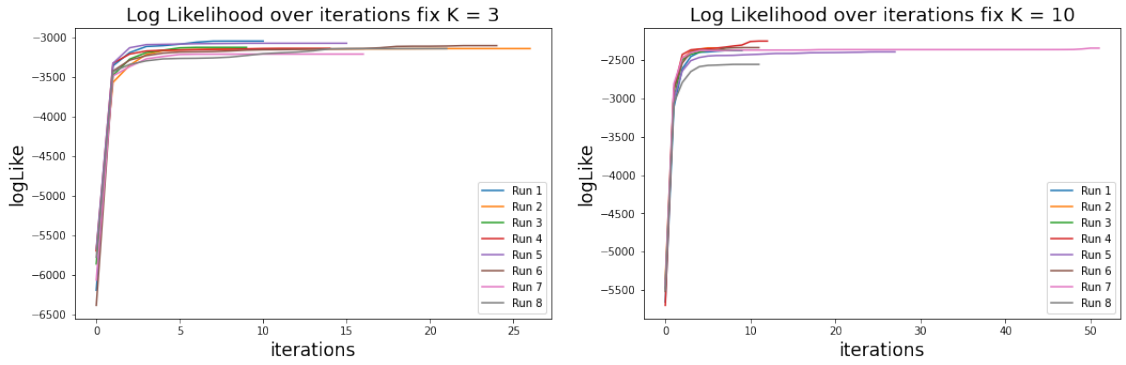
Figure 6: $P_{k=7}$

Figure 7: $P_{k=10}$

The plots above show different numbers 0,5,7. When I fix $k = 2$, I got different π_k for 7 and 0 or 0 and 5 at different time. But some of these π_1 and π_2 are similar. When I fix $k = 10$, I got π_k for many different combinations among 0,5,7.

- Comment on how well the algorithm works, whether it finds good clusters (look at the cluster means and responsibilities and try to interpret them), and how you might improve the model.

A: When I print each row of the responsibilities, I found that there is only one element with value one and the others with value zero. Hence, we can see this algorithm well assigns each image to only one cluster. From the plots below, we see the log likelihoods with 8 runs for $k = 3$ are much smaller than that for $k = 10$.



When k is small, the algorithm works well. When k is big, the algorithm shows bad performance. We got the answer which has too many repeated handwritten numbers. k is obviously more than actural number of distributions when k is big.
We can improve our model by collecting some prior information about our model. If we have known the rough number of k, it will imporve the acuracy of model and save time.

5. Decrypting Messages with MCMC.

(a) Give formulae for the ML estimates of these probabilities as functions of the counts of numbers of occurrences of symbols and pairs of symbols. Compute the estimated probabilities. Report the values as a table.

$S_1, S_2, \dots S_n$ where S_i are symbols as a Markov Chain

$$p(S_1 S_2 \dots S_n) = p(S_1) \cdot \prod_{i=2}^{n} p(S_i | S_{i-1})$$

$$\log(p(S_1 S_2 \dots S_n)) = \log(S_1) + \sum_{i=2}^{n} \log(p(S_i | S_{i-1}))$$

Let N be the counts of numbers of occurrences of symbols and pairs of symbols.

$\sum_{\beta} \psi(\alpha, \beta) = 1$. Use Lagrange multiplier λ, we have

$$\frac{\partial \log(p(S_1 S_2 \dots S_n)) + \sum \lambda (1 - \sum \psi(\alpha, \beta))}{\partial \psi(\alpha, \beta)} = \frac{N(\alpha\beta)}{\psi(\alpha, \beta)} - \lambda = 0$$

$$\Rightarrow \sum_{\beta} N(\alpha, \beta) = \sum_{\beta} \psi(\alpha, \beta) \cdot \lambda$$

$$\Rightarrow N(\alpha) = \lambda$$

$$\Rightarrow \psi(\alpha, \beta) = \frac{N(\alpha\beta)}{\lambda} = \frac{N(\alpha\beta)}{N(\alpha)}$$

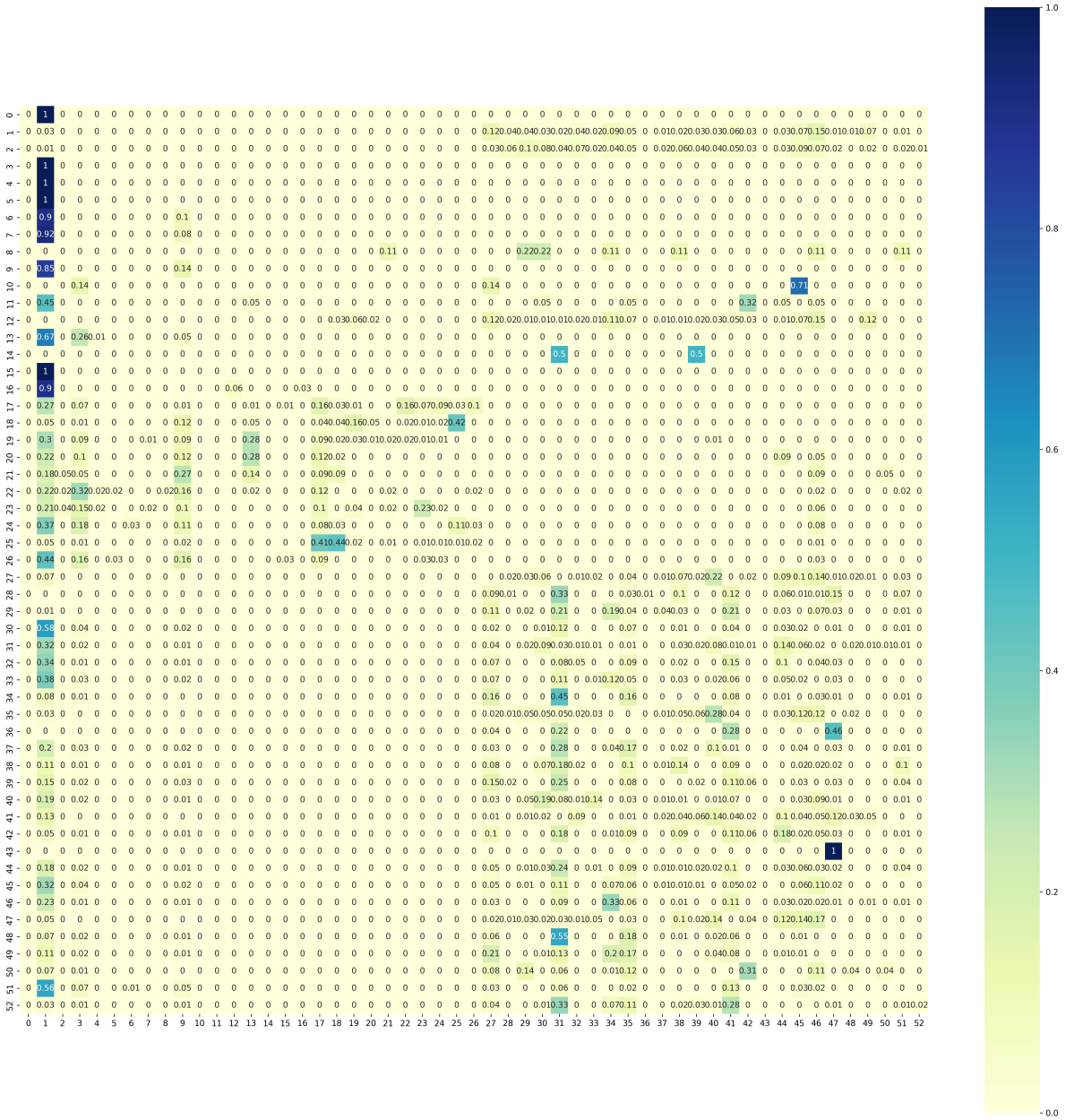


Figure 8: Estimated probabilities(two decimal places) with size 53×53

(b) Are the latent variables $\sigma(s)$ for different symbols s independent? Let $e_1 e_2 \dots e_n$ be an encrypted English text. Write down the joint probability of $e_1 e_2 \dots e_n$ given σ.

No, they are not independent. Since the mapping is one-to-one, once we select the previous $\sigma(s)$, the next one will not choose the previous symbol again. Hence, the probability of latent variables are dependent.

$e_1 e_2 \dots e_n$ encrypted text

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

(c) How does the proposal probability $S(\sigma \to \sigma')$ depend on the permutations σ and σ'? What is the MH acceptance probability for a given proposal?

Choosing two random s and s' among n characters

proposal probability: $S(\sigma \to \sigma') = \frac{1}{C_n^2} = \frac{1}{\binom{n}{2}}$

Since the proposal probability is symmetric, we have $S(\sigma \to \sigma') = S(\sigma' \to \sigma)$

$$\alpha(\sigma'|\sigma) = \min\left\{1, \frac{S(\sigma' \to \sigma) P(\sigma')}{S(\sigma \to \sigma') P(\sigma)}\right\}$$

$$= \min\left\{1, \frac{P(\sigma')}{P(\sigma)}\right\}$$

$$= \min\left\{1, \frac{P(\sigma'(e_1)) \prod_{i=2}^{n} P(\sigma'(e_i)|\sigma'(e_{i-1}))}{P(\sigma(e_1)) \prod_{i=2}^{n} P(\sigma(e_i)|\sigma(e_{i-1}))}\right\} \quad \to \psi(\alpha, \beta)$$

$$= \min\left\{1, \frac{P(e_1|\sigma')}{P(e_1|\sigma)} \prod_{i=2}^{n} \frac{\psi(\sigma'(e_i), \sigma'(e_{i-1}))}{\psi(\sigma(e_i), \sigma(e_{i-1}))}\right\}$$

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

$$= P(s_1) \cdot \prod_{i=2}^{n} P(s_i | s_{i-1})$$

(d) The general idea for this question: To decrypt Messages, we want to find the relation between each two symbols. To learn this, we use a book to calculate the probability between two adjacent symbols, which helps to simulate the occurrences of letters in actual text. We get a transition matrix.

We can see the 53 symbols as the states and use MCMC to decrypt. Then we randomly choose two symbols as the proposal. We find the accept rate by calculting the loglike. If it statisfies, we accept the proposal and transfer the letter. After many times training, the message decrypted should converge to a readable text. But the code might converge to a wrong text in practical. Hence, I tried many times to get the text. The detailed code and results see in appendix Q5. I found if I can get the correct

symbol for 'space', the code can easily decrypt the message, becasue 'space' can break the text into different words, which is easier to decrypt.

(e) Note that some $\psi(\alpha, \beta)$ values may be zero. Does this affect the ergodicity of the chain?
A: Yes, it does. The zeros will affect the ergodicity of the chain. For an ergodic chain, we need both irreducibility(non-zero prob.) and aperiodicity(non-zero diagonal).
To solve this, we can add a tiny value to each element of the transition matrix.

(f) Would symbol probabilities alone (rather than transitions) be sufficient?
A: Symbol probabilities alone is not sufficient. Because words are many combinations of letters, the relation between letters will provide useful information about the text.

If we used a second order Markov chain for English text, what problems might we encounter?
A: If we used a second order Markov chain, our transition matrix becomes a 3D matrix, which is really complexed and hard to use.

Will it work if the encryption scheme allows two symbols to be mapped to the same encrypted value?
A: No, it does not work. The map has to be one-to-one function otherwise we will fail to get the result. The process of calculating joint probability, MCMC and HM requires bijection mapping.

Would it work for Chinese with > 10000 symbols?
A: No, it does not. There are too many symbols which leads to a really big transition matrix and most entries in this matrix will be zero or extremely tiny value.

7. Optimization.

(a) Find the local extrema of the function $f(x, y) := x + 2y$ subject to the constraint $y^2 + xy = 1$.

$f(x,y) = x + 2y \quad g(x,y) = y^2 + xy = 1$

$\nabla f = \lambda \nabla g \Rightarrow \begin{cases} 1 = \lambda y & ① \\ 2 = \lambda(2y + x) & ② \\ y^2 + xy = 1 & ③ \end{cases}$

From ①, we get $y = \frac{1}{\lambda} \quad \lambda \neq 0$

From ②, we get $2 = \lambda(2 \cdot \frac{1}{\lambda} + x) \Leftrightarrow 2 + \lambda x = 2 \Rightarrow \lambda = 0$ or $x = 0$ (make no sense)

From ③, we get $y^2 = 1 \Rightarrow y = \pm 1$

We find our local extrema at $x = 0, y = \pm 1$.

$f(0,1) = 0 + 2 \times 1 = 2 \quad f(0,-1) = 0 + 2 \times (-1) = -2$

(b) Suppose we have a numerical routine to evaluate the exponential function $exp(x)$. How can we compute the function $ln(a)$, for a given $a \in \mathbb{R}_+$, using Newton's method?

$x = \ln(a) \Rightarrow e^x = a$

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

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

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

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

Check the convergence of Newton's algorithm.

The result converges iff $f(x) \cdot f''(x) < (f'(x))^2$

$(e^x - a)(e^x) = e^{2x} - a \cdot e^x < e^{2x} \quad (a > 0)$

Hence, it converges.

2 Bonus Questions

4. LGSSMs, EM and SSID.

(a) Run the function *ssm_kalman.py* that we have provided(or a re-implementation in your favourite language if you prefer) on the training data. Explain the behaviour of Y and V in both cases (and the differences between them).

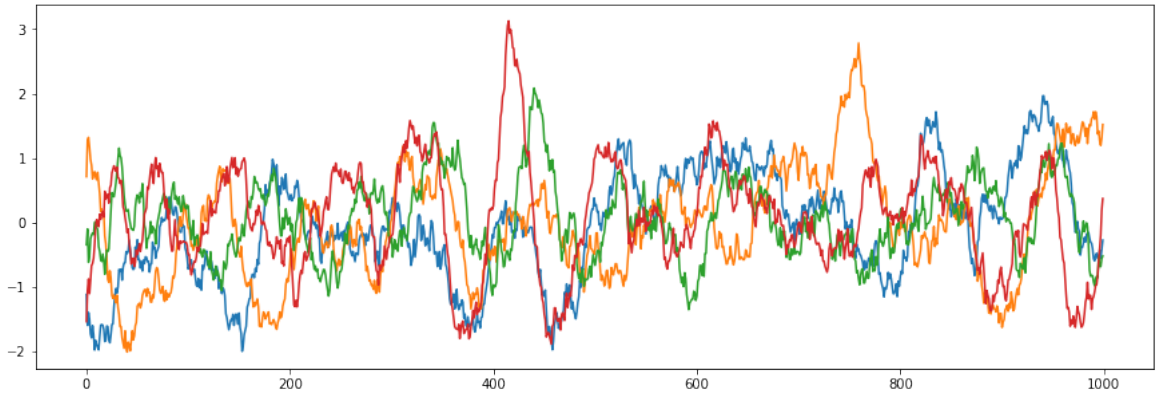


Figure 9: Kalman Filter

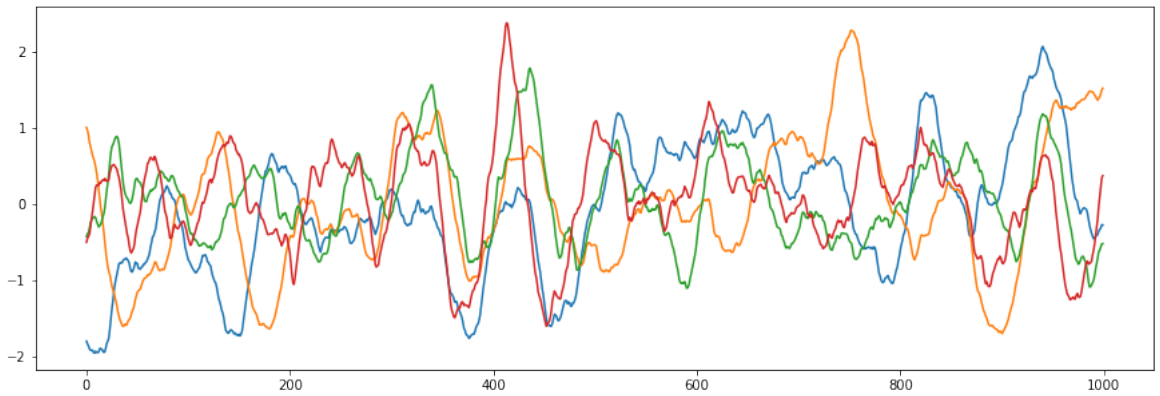


Figure 10: Kalman Smooth

The two plots above show the posterior mean estimates for the Kalman Filter and the Kalman Smooth. They have the similar trends but the Smooth plot looks more smooth.

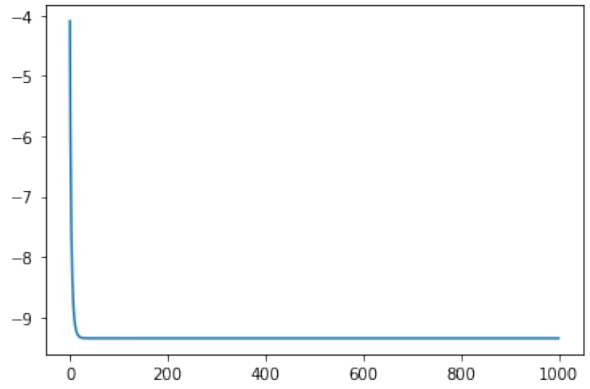


Figure 11: logdet(V) for Kalman Filter

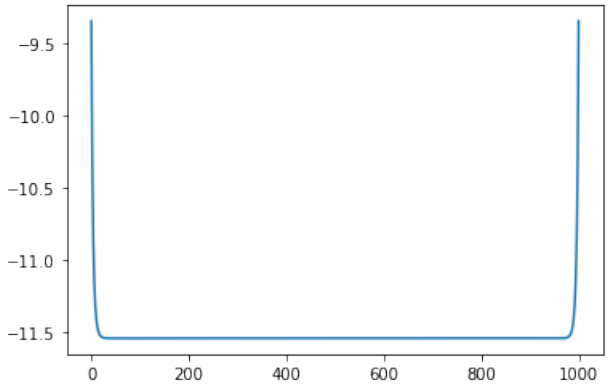
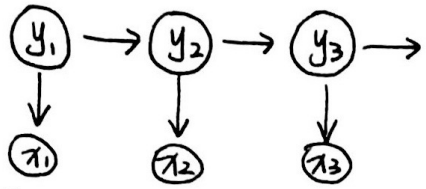


Figure 12: logdet(V) for Kalman Smooth

The two plots above show the log determinant of posterior variances on y_t. The Kalman Filter plot converges rapidly. The Kalman Smooth plot also converges rapidly but it suddenly grows up at the end. Generally, the values of logdet(cov) for Kalman Filter is larger than that for Kalman Smooth.
The Kalman smoothing seems like 'enhanced' version of the Kalman filter. The Kalman filter only uses information $X_{[0:t]}$. The Kalman filter uses the same information as Kalman filter($X_{[0:t]}$) and also use $X_{[t:T]}$. Hence, Kalman smooth gets more information($X_{[0:T]}$) and reduces the fluctuation of estimation results.

(b) Write a function to learn the parameters A, Q, C and R using EM (we will assume that the distribution on the first state is known a priori). The M-step update equations for A and C were derived in lecture.



Parameters: $\theta = \{y_0, Q_0, A, Q, R, C\}$

Free energy:

$$F(q,\theta) = \int dy_{1:T}\, q(y_{1:T})\left(\log P(x_{1:T}, y_{1:T}|\theta) - \log q(y_{1:T})\right)$$

$$p(y, x|\theta) = p(y_1)\cdot p(x_1|y_1)\prod_{t=2}^{T} p(y_t|y_{t-1})\,p(x_t|y_t)$$

$$R_{new} = \arg\max_R \left\langle \sum_t \log p(x_t|y_t)\right\rangle_q$$

$$= \arg\max_R \left\langle -\tfrac{1}{2}\sum_t (x_t - Cy_t)^T R^{-1}(x_t - Cy_t) + \sum_t \log(|R|^{-\frac{1}{2}})\right\rangle_q + \text{const}$$

$$= \arg\max_R \left\{ -\tfrac{1}{2}\sum_t x_t^T R^{-1} x_t - 2x_t R^{-1} C\langle y_t\rangle + \langle y_t^T C^T R^{-1} C y_t\rangle + \tfrac{T}{2}\log(|R^{-1}|)\right\}$$

$\left(\text{Tr}\left[\sum_t x_t^T R^{-1} C\langle y_t^T\rangle\right] - \frac{1}{2}\text{Tr}\left[\sum_t R^{-1} C\langle y_t y_t^T\rangle\right]\right)$

$$\frac{\partial f(R^{-1})}{\partial R^{-1}} = -\tfrac{1}{2}\sum_t x_t^T x_t + \tfrac{T}{2}R + \left(\sum_t x_t\langle y_t^T\rangle\right)C^T - \tfrac{1}{2}C\left(\sum_t \langle y_t y_t^T\rangle\right)C^T = 0$$

$$C_{new} = \left(\sum_t x_t\langle y_t\rangle^T\right)\left(\sum_t \langle y_t y_t^T\rangle\right)^{-1} \iff \sum_t \langle y_t y_t^T\rangle = \left(\sum_t x_t\langle y_t\rangle^T\right)\cdot C_{new}^{-1}$$

$$\frac{\partial f(R^{-1})}{\partial R^{-1}} = -\tfrac{1}{2}\sum_t x_t^T x_t + \tfrac{T}{2}R + \left(\sum_t x_t\langle y_t^T\rangle\right)C_{new}^T - \tfrac{1}{2}\,C_{new}\cancel{C_{new}^{-1}}\left(\sum_t x_t\langle y_t\rangle^T\right)C_{new}^T$$

$$= -\tfrac{1}{2}\sum_t x_t^T x_t + \tfrac{T}{2}R + \tfrac{1}{2}\left(\sum_t x_t\langle y_t\rangle^T\right)C_{new}^T$$

$$= 0$$

$$R_{new} = \frac{1}{T}\left[\sum_t x_t^T x_t - \left(\sum_t x_t\langle y_t\rangle^T\right)C_{new}^T\right]$$

$$Q_{new} = \arg\max_Q \left\langle \sum_t \log P(y_{t+1}|y_t)\right\rangle_q$$

$$= \arg\max_Q \left\langle -\tfrac{1}{2}\sum_t (y_{t+1} - Ay_t)^T Q^{-1}(y_{t+1} - Ay_t) + \sum_t \log(|Q|)^{-\frac{1}{2}}\right\rangle_q + \text{const}$$

$$= \arg\max_Q \left\{ -\tfrac{1}{2}\sum_t y_{t+1}^T Q^{-1} y_{t+1} - 2\langle y_{t+1}^T Q^{-1} A y_t\rangle + \langle y_t^T A^T Q^{-1} A y_t\rangle + \tfrac{T}{2}\log|Q^{-1}|\right\}$$

$$\frac{\partial f(Q^{-1})}{\partial Q^{-1}} = -\tfrac{1}{2}\sum_t \langle y_{t+1}^T y_{t+1}\rangle + \tfrac{T}{2}Q + \left(\sum_t \langle y_{t+1} y_t^T\rangle_q A^T - \tfrac{1}{2}A\sum_t \langle y_t y_t^T\rangle_q A^T = 0\right.$$

$$A_{new} = \left(\sum_t \langle y_{t+1} y_t^T\rangle\right)\left(\sum_t \langle y_t y_t^T\rangle\right)^{-1} \Rightarrow \sum_t \langle y_t y_t^T\rangle = A_{new}^{-1}\left(\sum_t \langle y_{t+1} y_t^T\rangle\right)$$

$$\frac{\partial f(R^{-1})}{\partial R^{-1}} = -\tfrac{1}{2}\sum_t \langle y_{t+1}^T y_{t+1}\rangle + \tfrac{T}{2}Q + \left(\sum_t \langle y_{t+1} y_t^T\rangle\right)A_{new}^T - \tfrac{1}{2}\,A_{new}\cancel{A_{new}^{-1}}\left(\sum_t \langle y_{t+1} y_t^T\rangle\right)A_{new}^T$$

$$= -\tfrac{1}{2}\sum_t \langle y_{t+1}^T y_{t+1}\rangle + \tfrac{T}{2}Q + \tfrac{1}{2}\left(\sum_t \langle y_{t+1} y_t^T\rangle\right)A_{new}^T$$

$$= 0$$

$$Q_{new} = \frac{1}{T}\left[\sum_t \langle y_{t+1}^T y_{t+1}\rangle - \left(\sum_t \langle y_{t+1} y_t^T\rangle\right)A_{new}^T\right] = \frac{1}{T-1}\left[\sum_{t=2}^{T} \langle y_t^T y_t\rangle - \left[\sum_{t=2}^{T} \langle y_t y_{t-1}^T\rangle\right]A_{new}^T\right]$$

8. Eigenvalues as solutions of an optimization problem.

(a) Use the extreme value theorem of calculus (recall: a continuous function on a compact domain attains its maximum and minimum) to show that $sup_{x \in R^n} = R_A(x)$ is attained.

A symmetric $n \times n$

$q_A(x) = x^T \cdot A \cdot x \qquad R_A(x) = \frac{x^T A x}{x^T x} = \frac{q_A(x)}{\|x\|^2} \quad x \in \mathbb{R}^n$

Let $y = \frac{x}{\|x\|}$. $\quad R_A(y) = \left(\frac{x}{\|x\|}\right)^T \cdot A \cdot \left(\frac{x}{\|x\|}\right) = y^T A y$

$y \in \mathbb{R}^n$ and $\|y\| = 1$ The unit sphere $S = \{y \in \mathbb{R}^n \mid \|y\| = 1\}$ is compact.

$\|y\| = \sqrt{y^2} = \left(\frac{\sum x_n^2}{\|x\|^2}\right)^{\frac{1}{2}} = 1$

Now, it's the same as to show $\sup_{y \in \mathbb{R}^n} R_A(y)$ is attained.

Since we know a continuous function on a compact domain attains its maximum and minimum, $\sup_{y \in \mathbb{R}^n} R_A(y)$ is attained as S is compact.

(b) Show that $R_A(x) \leq \lambda_1$

$\lambda_1 \geq \lambda_2 \geq \dots \lambda_n$ eigenvalues

$f_1 \; f_2 \; \dots \; f_n$ eigenvectors form an ONB

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

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

$$= \frac{\left(\sum_{i=1}^{n} (f_i x) f_i\right)^T \cdot A \cdot \left(\sum_{i=1}^{n} (f_i^T x) f_i\right)}{\left(\sum_{i=1}^{n} (f_i x) f_i\right)^T \cdot \left(\sum_{i=1}^{n} (f_i^T x) f_i\right)}$$

$$= \frac{\left(\sum_{i=1}^{n} (f_i x) f_i\right)^T \left(\sum_{i=1}^{n} (f_i^T x) A f_i\right)}{\sum_{i=1}^{n} \left((f_i x) f_i\right)^T \left((f_i^T x) f_i\right)} \qquad A f_i = \lambda_i f_i$$

$$= \frac{\left(\sum_{i=1}^{n} (f_i x) f_i\right)^T \cdot \left(\sum_{i=1}^{n} (f_i^T x) \lambda_i f_i\right)}{\sum_{i=1}^{n} f_i^T (f_i x)^T (f_i^T x) f_i}$$

$$= \frac{\sum_{i=1}^{n} \lambda_i \left((f_i x) f_i\right)^T \left((f_i^T x) f_i\right)}{\sum_{i=1}^{n} f_i^T \cdot x^T \cdot f_i^T \cdot f_i^T \cdot x \cdot f_i}$$

$$= \frac{\sum_{i=1}^{n} \lambda_i \left(f_i^T \cdot x^T \cdot f_i^T \cdot f_i^T \cdot x \cdot f_i\right)}{\sum_{i=1}^{n} f_i^T \cdot x^T \cdot f_i^T \cdot f_i^T \cdot x \cdot f_i} \qquad \lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_n$$

$$\leq \lambda_1 \left(\frac{\sum_{i=1}^{n} \dots}{\sum_{i=1}^{n} \dots}\right) = \lambda_1$$

$\left(\frac{\sum_{i=1}^{n} \dots}{\sum_{i=1}^{n} \dots}\right) = 1$

(c) Show that, if $x \in R^n$ is not contained in $\text{span}\{\xi_1, \dots, \xi_k\}$, then $R_A(x) < \lambda_1$.

$R_A(f_1) = \lambda_1$

$f_1, f_2, \dots f_k \qquad k \leq n$

$x \in \mathbb{R}^n$ is not contained in $\text{span}\{f_1, \dots, f_k\}$

$\Rightarrow x = c_{k+1} f_{k+1} + c_{k+2} f_{k+2} + \dots + c_n f_n$

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

$$= \frac{\sum_{i=k+1}^{n} \lambda_i \left(f_i^T \cdot x^T \cdot f_i^T \cdot f_i^T \cdot x \cdot f_i\right)}{\sum_{i=k+1}^{n} \left(f_i^T x^T f_i^T f_i^T \cdot x \cdot f_i\right)} \qquad \lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_{k+1} \geq \dots \geq \lambda_n$$

$\leq \lambda_{k+1} < \lambda_1$

Appendix

Q1

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

def Max_likelihood(X):
    D,N = X.shape
    p = np.zeros(D)
    for d in range(D):
        sum_xd = 0
        for n in range(N):
            sum_xd += X[d,n]
        p[d] = sum_xd/D

    return p

def Max_APosteriori(X, alpha, beta):
    D,N = X.shape
    p = np.zeros(D)
    for d in range(D):
        sum_xd = 0
        for n in range(N):
            sum_xd += X[d,n]
        p[d] = (sum_xd + alpha - 1) / (alpha + beta + N - 2)

    return p

Y = np.loadtxt('binarydigits.txt')
X = Y.T

# Q1(d)
a = Max_likelihood(X)
plt.figure()
plt.imshow(np.reshape(a, (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.savefig('Plot_1(d).png', bbox_inches = 'tight')

# Q1(e)
b = Max_APosteriori(X, 3, 3)
plt.figure()
plt.imshow(np.reshape(b, (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.savefig('Plot_1(e).png', bbox_inches = 'tight')
```

Q2

```
from scipy.special import betaln

Y = np.loadtxt('binarydigits.txt')
X = Y.T
D,N = X.shape

# (DN)*log(1/2) + log(1/3)
log_p_Ma = (D*N)*np.log(1/2)+np.log(1/3)
print('The log posterior probabilities for model A is '+ str(log_p_Ma))

# log(B(sum(X)+1,sum(1-X)+1)) + log(1/3)
log_p_Mb = betaln(np.sum(X)+1,np.sum(1-X)+1) + np.log(1/3)
print('The log posterior probabilities for model B is '+ str(log_p_Mb))

# sum(log(B(sum(x_d)+1,sum(1-x_d)+1))) + log(1/3)
Log_p_Mc = []
for d in range(D):
    log_p_Mc = betaln(np.sum(X[d,:])+1,N - np.sum(X[d,:])+1)
    Log_p_Mc.append(log_p_Mc)

log_p_Mc = np.sum(Log_p_Mc)+np.log(1/3)
print('The log posterior probabilities for model C is '+ str(log_p_Mc))
```

Q3

```
def get_Multi_Ber_logLike(X,K,P,W):
    # X: given data.transpose(otherwise we have shape (N,D))
    # K: K multivariate Bernoullis
    # P: Bernoulli parameter vectors into a matrix P
    # W: weights(mixing proportions pi)

    D, N = X.shape
    loglike = 0

    for n in range(N):
        Like = []
        for k in range(K):
            like = W[k] * np.prod((P[:,k]**X[:, n])*((1-P[:,k])**(1-X[:, n])))
            Like.append(like)

        loglike += np.log(np.sum(Like))

    return loglike


def E_step(X,P,W,K):
    D, N = X.shape
    R = np.zeros(shape=(K,N)) # responsibility
    for n in range(N):
        like = np.zeros(K)
        for k in range(K):
            like[k] = W[k] * np.prod((P[:,k]**X[:, n])*((1-P[:,k])**(1-X[:, n])))

        R[:, n] = like/sum(like)

    return R


def M_step(R,X,K):
    # R: responsibility
    D, N = X.shape
    P = np.zeros(shape=(D,K))
    W = np.zeros(K)
    for k in range(K):
        P[:,k] = np.sum(R[k,:]*X,1)/np.sum(R[k,:]) # sum(r_nk*x_d)/sum(r_nk)
        W[k] = np.sum(R[k,:])/ N # sum(r_nk)/N

    return P,W
```

```
def BerMixModel(Y,K,I):
    # Y: given data
    # K: K multivariate Bernoullis
    # I: interations

    X = Y.T
    D, N = X.shape

    # define inital values
    P = np.random.uniform(size=(D,K)) # Bernoulli parameter vectors into a matrix P
    w = np.random.uniform(size=K) # each pi
    pi = w/sum(w) # pi list
    R = np.zeros(shape=(K,N)) # responsibility
```

```
    # create an empty list
    # get inital loglike given above inital values
    # put it into the list
    logLike = []
    loglike = get_Multi_Ber_logLike(X,K,P,pi)
    logLike.append(loglike)

    # begin iterations
    for i in range(I):

        # E step
        # calculate responsibility R
        R = E_step(X,P,pi,K)

        # M step
        # update weight, parameters
        P,W = M_step(R,X,K)

        # update loglike and the list
        loglike_update = get_Multi_Ber_logLike(X,K,P,W)
        logLike.append(loglike_update)

        # stopping critera
        if(abs(logLike[i] - logLike[i+1]) < 0.0001):
            break
        else:
            continue

    return R, P, pi, logLike
```

```
# Q3(d)
Y = np.loadtxt('binarydigits.txt')
List = [2,3,4,7,10]
plt.figure(figsize = (8,5))

for l in List:
    R, P, pi, loglike = BerMixModel(Y,K=l,I=20)
    plt.plot(loglike, label= 'k = ' + str(l))
    print('The parameters $pi_k$ for k = '+ str(l)+ ' is ' + str(pi))

plt.legend()
plt.title('Log Likelihood over iterations', fontsize=18)
plt.xlabel('Iterations', fontsize=17)
plt.ylabel('logLike', fontsize=17)
plt.savefig('3(d).png', bbox_inches = 'tight')
```

```
# Q3(e)
plt.figure(figsize = (8,5))
for i in range(8):
    R, P, pi, loglike = BerMixModel(Y,K=3,I=100)
    plt.plot(loglike, label= 'Run ' + str(i+1))

plt.legend()
plt.title('Log Likelihood over iterations fix K = 3', fontsize=18)
plt.xlabel('iterations', fontsize=17)
plt.ylabel('logLike', fontsize=17)
plt.savefig('3(e)(i).png', bbox_inches = 'tight')

plt.figure(figsize = (8,5))
for i in range(8):
    R, P, pi, loglike = BerMixModel(Y,K=10,I=100)
    plt.plot(loglike, label= 'Run ' + str(i+1))

plt.legend()
plt.title('Log Likelihood over iterations fix K = 10', fontsize=18)
plt.xlabel('iterations', fontsize=17)
plt.ylabel('logLike', fontsize=17)
plt.savefig('3(e)(ii).png', bbox_inches = 'tight')
```

```
Y = np.loadtxt('binarydigits.txt')
K = 2
I = 20
i = 0
plt.figure(figsize = (2,2))

R, P, pi, loglike = BerMixModel(Y,K,I)
plt.subplot(1,2,1)

plt.imshow(np.reshape(P[:,0], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')

plt.subplot(1,2,2)
plt.imshow(np.reshape(P[:,1], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')

plt.savefig('3(e)(k=2).png', bbox_inches = 'tight')
```

```
K = 3
plt.figure(figsize = (3,3))
R, P, pi, loglike = BerMixModel(Y,K,I)
plt.subplot(1,3,1)
plt.imshow(np.reshape(P[:,0], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,3,2)
plt.imshow(np.reshape(P[:,1], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,3,3)
plt.imshow(np.reshape(P[:,2], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.savefig('3(e)(k=3).png', bbox_inches = 'tight')
```

```
K = 4
plt.figure(figsize = (4,4))
R, P, pi, loglike = BerMixModel(Y,K,I)
plt.subplot(1,4,1)
plt.imshow(np.reshape(P[:,0], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,4,2)
plt.imshow(np.reshape(P[:,1], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,4,3)
plt.imshow(np.reshape(P[:,2], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,4,4)
plt.imshow(np.reshape(P[:,3], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.savefig('3(e)(k=4).png', bbox_inches = 'tight')
```

```
K = 7
plt.figure(figsize = (7,7))
R, P, pi, loglike = BerMixModel(Y,K,I)
plt.subplot(1,7,1)
plt.imshow(np.reshape(P[:,0], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,7,2)
plt.imshow(np.reshape(P[:,1], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,7,3)
plt.imshow(np.reshape(P[:,2], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,7,4)
plt.imshow(np.reshape(P[:,3], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,7,5)
plt.imshow(np.reshape(P[:,4], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,7,6)
plt.imshow(np.reshape(P[:,5], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,7,7)
plt.imshow(np.reshape(P[:,6], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.savefig('3(e)(k=7).png', bbox_inches = 'tight')
```

```
K = 10
plt.figure(figsize = (10,10))
R, P, pi, loglike = BerMixModel(Y,K,I)
plt.subplot(1,10,1)
plt.imshow(np.reshape(P[:,0], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,10,2)
plt.imshow(np.reshape(P[:,1], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,10,3)
plt.imshow(np.reshape(P[:,2], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,10,4)
plt.imshow(np.reshape(P[:,3], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,10,5)
plt.imshow(np.reshape(P[:,4], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,10,6)
plt.imshow(np.reshape(P[:,5], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
```

```
plt.subplot(1,10,7)
plt.imshow(np.reshape(P[:,6], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,10,8)
plt.imshow(np.reshape(P[:,7], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,10,9)
plt.imshow(np.reshape(P[:,8], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.subplot(1,10,10)
plt.imshow(np.reshape(P[:,9], (8,8)),interpolation="None",cmap='gray')
plt.axis('off')
plt.savefig('3(e)(k=10).png', bbox_inches = 'tight')
```

```
print(R[:,0])
print(R[:,1])
print(R[:,2])
print(R[:,3])
print(R[:,4])
```

Q4

```
# Q4(a)
# logdet = @(A)(2*sum(log(diag(chol(A)))))
def logdet(A):
    logdet= 2*np.sum(np.log(np.diag(np.linalg.cholesky(A))))
    return logdet

X_train = np.loadtxt('ssm_spins.txt')

A = 0.99 * np.array([[np.cos(2*np.pi/180), - np.sin(2*np.pi/180), 0, 0],
                     [np.sin(2*np.pi/180), np.cos(2*np.pi/180),0,0],
                     [0,0,np.cos(2*np.pi/90), - np.sin(2*np.pi/90)],
                     [0,0,np.sin(2*np.pi/90), np.cos(2*np.pi/90)]])

AA= np.dot(A,A.T)
Q = np.identity(len(AA)) - AA

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

R = np.identity(len(C))

# define initalv value
Q_0 = np.identity(len(A))
# multivariate normal y ~ N(Ay,Q)
y_0 = np.random.multivariate_normal(np.zeros(len(A)),np.identity(len(A)))
```

```
y_hat, V_hat, V_joint, likelihood = run_ssm_kalman(X_train.T, y_0, Q_0, A, Q, C, R, mode='filt')
V = []
for i in V_hat:
    v = logdet(i)
    V.append(v)

plt.figure(figsize=(15,5))
plt.plot(y_hat.T)
plt.savefig('4(a)(11).png', bbox_inches = 'tight')


plt.figure()
plt.plot(V)
plt.savefig('4(a)(12).png', bbox_inches = 'tight')
```

```
y_hat, V_hat, V_joint, likelihood = run_ssm_kalman(X_train.T, y_0, Q_0, A, Q, C, R, mode='smooth')
V = []
for i in V_hat:
    v = logdet(i)
    V.append(v)

plt.figure(figsize=(15,5))
plt.plot(y_hat.T)
plt.savefig('4(a)(21).png', bbox_inches = 'tight')

plt.figure()
plt.plot(V)
plt.savefig('4(a)(22).png', bbox_inches = 'tight')
```

Q5

```
import numpy as np
from matplotlib import pyplot as plt
import random
import seaborn as sns
import pandas as pd
import math

message = open('message.txt','r')
message = message.read()

symbols = open('symbols.txt','r')
symbols = symbols.read().replace('\n','')

war_and_peace = open('war_and_peace.txt', 'r')
war_and_peace = war_and_peace.read().replace('\n','')
war_and_peace = war_and_peace.lower() #return the string to lowercase

# clean useless characters and blank space
replace_ch = ['~','@','#','$','%','^','&','_','+','`','{','}','|','<','>']
for ch in replace_ch:
      war_and_peace = war_and_peace.replace(ch, '')

replace_space = ['  ','   ','    ','     ','      ','       ','        ']
for sp in replace_space:
      war_and_peace = war_and_peace.replace(sp, ' ')


def trans_mat(text,symbols):
    n = len(symbols)
    l = len(text)
    trans_mat = np.zeros((n,n))

    for i in range(l):
        # check if the characters at position i and i-1 appear in symbols
        if (symbols.count(text[i]) > 0) and (symbols.count(text[i-1]) > 0):
            index_i = symbols.index(text[i-1])
            index_j = symbols.index(text[i])
            trans_mat[index_i,index_j] += 1
        else:
        # if not, do nothing
            continue

    r_sum = np.sum(trans_mat,axis = 1) # calculate the sum for each row
    Equ_mat = np.zeros((n,n))
    for i in range(n):
        # compute the propotion by dividing r_sum
        Equ_mat[i,:] = trans_mat[i,:]/r_sum[i]

    return Equ_mat

# Q5(a)
trans_matrix = trans_mat(war_and_peace,symbols)
plt.figure(figsize = (25,25))
p1 = sns.heatmap(pd.DataFrame(np.round(trans_matrix,2)),
                 annot=True,xticklabels= True, yticklabels= True, square=True, cmap="YlGnBu")
s1 = p1.get_figure()
s1.savefig('HeatMap.jpg',dpi=300,bbox_inches='tight')
```

```
def RandSwap(symbols):
    l = len(symbols)
    symbols = list(symbols)
    # random choose two numbers from 0 to l-1
    s1 = random.randint(0,l-1)
    s2 = random.randint(0,l-1)
    if symbols[s1] == symbols[s2]:
        return RandSwap(symbols) # recursion
    else:
        new_symbols = symbols
        # swap the corresponding encrypted symbols
        new_symbols[s1],new_symbols[s2] = new_symbols[s2],new_symbols[s1]
        return new_symbols
```

```
# for each character in text, find its position in symbols,
# and then find the mapping character in key
def get_decrypt_message(text,symbols,key):
    new_text = []
    l = len(text)
    for i in range(l):
        new_text.append(key[symbols.index(text[i])])
    return ''.join(new_text) # return a string
```

```
# prod of p(e_i\e_i-1) as score
# the higher the score, the higher prob it will transfer
def log_score(text,symbols,key,matrix):
    new_text = get_decrypt_message(text,symbols,key)
    l = len(text)
    #like = 1
    loglike = 0
    for i in range(l):
        # too tiny, we use log score instead of score
        #like *= matrix[symbols.index(new_text[i-1]),symbols.index(new_text[i])]

        # log(lik1*lik2*...) = sum(log(lik1)+log(lik2)+...)
        loglike += np.log(matrix[symbols.index(new_text[i-1]),symbols.index(new_text[i])])
    return loglike
```

```
def MCMC_HM(text,matrix,symbols,n):
    key = list(symbols)
    for i in range(n):
        current_key = key
        proposed_key = RandSwap(key)
        current_prob = log_score(message,symbols,current_key,matrix)
        proposed_prob = log_score(message,symbols,proposed_key,matrix)
        A = min(1,np.exp(proposed_prob-current_prob)) # accept probability
        U = random.uniform(0,1)
        # if proposed_prob increases, accept proposal;
        # if u <= accept probability, accept proposal
        if  (U <= A)  or (current_prob < proposed_prob):
            key = proposed_key
        if i%100 == 0:
            print('iteration',i,':')
            print(get_decrypt_message(text,symbols,key)[0:60])
            print('')
    return key
```

```
# Q5(d)
trans_matrix = trans_matrix + 0.0001 # restore ergodicity
new_key = MCMC_HM(message, trans_matrix, symbols,20000)
```

```
iteration 0 :
[:p1lpl]x:r?!pw:np1]!?pmx;:?!w.;?pl?w!gp1lp/ws-?!prwm?p1?pg]

iteration 100 :
y:polplhi:sdapw: pohadpmi.:dawk.dpldwagpolp,wr-dapswmdpodpgh

iteration 200 :
eoayhahuiobdpamo ayupdalisodpmwsdahdmpgayha,mr-dpabmldaydagu

iteration 300 :
eo yh humocdn ioa yund lmpodniwpd hding yh ,ir-dn cild yd gu

iteration 400 :
es ph hous,da isr poad lumsdaiwmd hdiag ph fin-da ,ild pd go
```

```
iteration 500 :
es ph hous,dl isa pold rumsdli-md hdilg ph finwdl ,ird pd go

iteration 600 :
es ph hous,dl isa pold rumsdli-md hdilg ph fintdl ,ird pd go

iteration 700 :
en fh houn,dl ina fold rugndli-gd hdilm fh pistdl ,ird fd mo

iteration 800 :
en th houn,dl ina told rugndli-gd hdilm th fispdl ,ird td mo

iteration 900 :
dn th houn,el ina tole rugneli-ge heilm th fiswel ,ire te mo

iteration 1000 :
an th houn,el ind tole mugneli-ge heilr th fiswel ,ime te ro

iteration 1100 :
an tw woun,el ind tole mugneli-ge weilr tw fishel ,ime te ro

iteration 1200 :
an tw wounpel ind tole mugneli-ge weilr tw fishel pime te ro

iteration 1300 :
an rw woungel ind role mupnelivpe weilt rw fishel gime re to

iteration 1400 :
an r. .oungel ind role mupnelivpe .eilt r. fishel gime re to

iteration 1500 :
an r. .oungel ind role mupnelivpe .eilt r. fishel gime re to

iteration 1600 :
an l. .ounger ind lore mupnerivpe .eirs l. fither gime le so

iteration 1700 :
an l. .ounger ind lore mupnerivpe .eirs l. fither gime le so

iteration 1800 :
an l. .ounger ind lore mupnerivpe .eirs l. fither gime le so

iteration 1900 :
an l. .ounger ind lore mupnerivpe .eirs l. fither gime le so

iteration 2000 :
an l. .ounger ind lore mupnerivpe .eirs l. fither gime le so

iteration 2100 :
an l. .ounger ind lore mupnerivpe .eirs l. fither gime le so

iteration 2200 :
an m. .ounger ind more lupnerivpe .eirs m. fither gile me so

iteration 2300 :
an m, ,ounger ind more lupnerivpe ,eirs m, fither gile me so

iteration 2400 :
an my younger ind more pulnerivle yeirs my fither gipe me so
```

```
iteration 2500 :
an my younger ind more pulnerivle yeirs my fither gipe me so

iteration 2600 :
an my younger ind more bulnerivle yeirs my fither gibe me so

iteration 2700 :
an my younger ind more bulnerivle yeirs my fither gibe me so

iteration 2800 :
an my younger ind more bulnerivle yeirs my fither gibe me so

iteration 2900 :
an my younger ind more vulnerible yeirs my fither give me so

iteration 3000 :
an my younger ind more vulnerible yeirs my fither give me so

iteration 3100 :
an my younger ind more vulnerible yeirs my fither give me so

iteration 3200 :
in my younger and more vulnerable years my father gave me so

iteration 3300 :
in my younger and more vulnerable years my father gave me so

iteration 3400 :
in my younger and more vulnerable years my father gave me so

iteration 3500 :
in my younger and more vulnerable years my father gave me so

iteration 3600 :
in my younger and more vulnerable years my father gave me so

iteration 3700 :
in my younger and more vulnerable years my father gave me so

iteration 3800 :
in my younger and more vulnerable years my father gave me so

iteration 3900 :
in my younger and more vulnerable years my father gave me so

iteration 4000 :
in my younger and more vulnerable years my father gave me so

iteration 4100 :
in my younger and more vulnerable years my father gave me so

iteration 4200 :
in my younger and more vulnerable years my father gave me so

iteration 4300 :
in my younger and more vulnerable years my father gave me so
```

The text did not change any more after iteration 3200.