EM for Binary Data.

Consider the data set of binary (black and white) images used in the previous assignment. Each image is arranged into a vector of pixels by concatenating the columns of pixels in the image. The data set has N images $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$ and each image has D pixels, where D is (number of rows $\times$ number of columns) in the image. For example, image $\mathbf{x}^{(n)}$ is a vector $(x_1^{(n)}, \ldots, x_D^{(n)})$ where $x_d^{(n)} \in \{0, 1\}$ for all $n \in \{1, \ldots, N\}$ and $d \in \{1, \ldots, D\}$.

(a) Write down the likelihood for a model consisting of a mixture of K multivariate Bernoulli distributions. Use the parameters $\pi_1, \ldots, \pi_K$ to denote the mixing proportions ($0 \leq \pi_k \leq 1; \sum_k \pi_k = 1$) and arrange the K Bernoulli parameter vectors into a matrix P with elements p_{kd} denoting the probability that pixel d takes value 1 under mixture component k. Assume the images are iid under the model, and that the pixels are independent of each other within each component distribution.

Just as we can for a mixture of Gaussians, we can formulate this mixture as a latent variable model, introducing a discrete hidden variable $s^{(n)} \in \{1, \ldots, K\}$ where $P(s^{(n)} = k|\boldsymbol{\pi}) = \pi_k$.

(b) Write down the expression for the responsibility of mixture component k for data vector $\mathbf{x}^{(n)}$, i.e. $r_{nk} \equiv P(s^{(n)} = k|\mathbf{x}^{(n)}, \boldsymbol{\pi}, \mathsf{P})$. This computation provides the E-step for an EM algorithm.

(c) Find the maximizing parameters for the expected log-joint

$$\underset{\boldsymbol{\pi}, \mathsf{P}}{\operatorname{argmax}} \left\langle \sum_n \log P(\mathbf{x}^{(n)}, s^{(n)}|\boldsymbol{\pi}, \mathsf{P}) \right\rangle_{q(\{s^{(n)}\})}$$

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

(d) Implement the EM algorithm for a mixture of K multivariate Bernoullis.

Your code should take as input the number K, a matrix X containing the data set, and a maximum number of iterations to run. The algorithm should run for that number of iterations or until the log likelihood converges (does not increase by more than a very small amount).

Hand in clearly commented code and a description of how the code works.

Run your algorithm on the data set for values of K in $\{2, 3, 4, 7, 10\}$. Report the log likelihoods obtained and display the parameters found.

[Hints: Although the implementation may seem simple enough given the equations, there are many numerical pitfalls (that are common in much of probabilistic learning). A few suggestions:

- Likelihoods can be very small; it is often better to work with log-likelihoods.

- You may still encounter numerical issues computing responsibilities. Consider scaling the numerator and denominator of the equation by a suitable constant while still in the log domain.
- It may also help to introduce (weak) priors on P and $\boldsymbol{\pi}$ and use EM to find the MAP estimates, rather than ML.
- If working in MATLAB, consider "vectorizing" your code, avoiding `for` loops where possible. This often [though not always] makes the code more readable and lets it run faster.
- It is always useful to plot the log-likelihood after every iteration. If it ever decreases (when implementing EM for ML) you have a bug! The same is true for the log-joint when implementing MAP.

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

(f) Run the algorithm a few times starting from randomly chosen initial conditions. Do you obtain the same solutions (up to permutation)? Does this depend on K?

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.

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

1. (a) $\mathcal{D} = \{\underset{\sim}{x}^{(1)}, \cdots, \underset{\sim}{x}^{(N)}\}$, $\underset{=}{X} = \begin{pmatrix} \vdots & \vdots & & \vdots \\ \underset{\sim}{x}^{(1)} & \underset{\sim}{x}^{(2)} & \cdots & \underset{\sim}{x}^{(N)} \\ \vdots & \vdots & & \vdots \end{pmatrix} \in \mathbb{R}^{D \times N}$ d: # of pixel, n: # of image.

$\underset{=}{P} = \begin{pmatrix} p_{11} & \cdots & p_{1D} \\ \vdots & & \vdots \\ p_{K1} & \cdots & p_{KD} \end{pmatrix} \in \mathbb{R}^{K \times D}$, $\underset{\sim}{p}_k = (p_{k1} \cdots p_{kD}) \in \mathbb{R}^{1 \times D}$

$\underset{\sim}{\pi} = (\pi_1, \cdots, \pi_K) \in \mathbb{R}^{1 \times K}$

$$p(\underset{\sim}{x}^{(n)} \mid \underset{\sim}{p}_k) = \prod_{d=1}^{D} p_{kd}^{x_d^{(n)}} \cdot (1 - p_{kd})^{(1 - x_d^{(n)})}.$$

since every image is a mixture of Bernulli

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

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

(b) $\gamma_{nk} = p(s^{(n)} = k \mid \underset{\sim}{x}^{(n)}, \underset{\sim}{\pi}, \underset{=}{P})$ (definition)

$$= \frac{p(s^{(n)} = k \mid \underset{\sim}{\pi}, \underset{=}{P}) \cdot p(\underset{\sim}{x}^{(n)} \mid s^{(n)} = k, \underset{\sim}{\pi}, \underset{=}{P})}{\sum_{j=1}^{K} p(s^{(n)} = j \mid \underset{\sim}{\pi}, \underset{=}{P}) \cdot p(\underset{\sim}{x}^{(n)} \mid s^{(n)} = j, \underset{\sim}{\pi}, \underset{=}{P})}$$

$$= \frac{\pi_k \cdot p(\underset{\sim}{x}^{(n)} \mid s^{(n)} = k, \underset{\sim}{\pi}, \underset{=}{P})}{\sum_{j=1}^{K} \pi_j \cdot p(\underset{\sim}{x}^{(n)} \mid s^{(n)} = j, \underset{\sim}{\pi}, \underset{=}{P})} = \frac{\pi_k \cdot \prod_{d=1}^{D} p_{kd}^{x_d^{(n)}} \cdot (1 - p_{kd})^{(1 - x_d^{(n)})}}{\sum_{j=1}^{K} \pi_j \prod_{d=1}^{D} p_{jd}^{x_j^{(n)}} \cdot (1 - p_{jd})^{(1 - x_j^{(n)})}}$$

(c) $E = \left\langle \sum_{n=1}^{N} \log p(\underset{\sim}{x}^{(n)}, s^{(n)} \mid \underset{\sim}{\pi}, \underset{=}{P}) \right\rangle_{q(\{s^{(n)}\})}$. π_k from definition

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

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

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

$$* \quad \frac{\partial E}{\partial \underset{\sim}{p}_k} = \sum_{n=1}^{N} \gamma_{nk} \left\{ \frac{\underset{\sim}{x}^{(n)}}{\underset{\sim}{p}_k} - \frac{(1 - \underset{\sim}{x}^{(n)})}{1 - \underset{\sim}{p}_k} \right\} = 0.$$

$$\Leftrightarrow \sum_n \gamma_{nk} \underset{\sim}{x}^{(n)} - \underset{\sim}{p}_k \sum_n \gamma_{nk} \underset{\sim}{x}^{(n)} = \underset{\sim}{p}_k \sum_n \gamma_{nk} - \underset{\sim}{p}_k \sum_n \gamma_{nk} \underset{\sim}{x}^{(n)}$$

$$\Leftrightarrow \underset{\sim}{p}_k = \frac{\sum_n \gamma_{nk} \underset{\sim}{x}^{(n)}}{\sum_n \gamma_{nk}} = \frac{1}{N_k} \sum_n \gamma_{nk} \cdot \underset{\sim}{x}^{(n)}$$

Lagrian multiplier for unconstrained optimisation

$$* \quad \frac{\partial}{\partial \pi_k} \left\{ \left\langle \sum_{n=1}^{N} \log p(\underset{\sim}{x}^{(n)}, s^{(n)} \mid \underset{\sim}{\pi}, \underset{=}{P}) \right\rangle_{q(\{s^{(n)}\})} + \lambda \left(\sum_{j=1}^{K} \pi_j - 1 \right) \right\} = 0$$

$$\Leftrightarrow \lambda = -N \text{ and } \pi_k = \frac{1}{N} \sum_n \gamma_{nk} = \frac{N_k}{N}.$$

(d) Body, run this in MATLAB

```
clear; clc; clf;load binarydigits.txt -ascii;
Y=binarydigits;
[N, D] = size(Y); % D = 64 (number of pixels), N = 100 (number of images)
X = Y'; % dataset X, with size(X) = [D N]
K = 10;  % chosen K value
% Random initialise pi and P, where pi is the vector of mixing cofficient;
% P is the matrix of Bernoulli paramaers
pi = rand(1,K);
pi = pi/sum(pi); % normalise and update pi, so that sum(pi) = 1
P = rand(K,D);
% Iteration
num_steps = 20;
log_likelihood_store = zeros(num_steps,1);
for i  = 1:num_steps
    R  = calculateResponsibilities(pi, P, X, K); % E step
    P  = calculateParameter(R,X,K);              % M step
    pi = calculateMixingCofficient(R,X,K);       % M step
    log_likelihood_store(i) = evluateLogLikelihood(R,P,pi,X,K); % M step
end
log_likelihood_store % print to check whether it converges
plot(log_likelihood_store)
```

```
function R = calculateResponsibilities(pi, P, X, K)
% Calculate responsibility matrix R
% Input:
% pi: mixing cofficient vector
% P:  parameter matrix
% X:  date
% K:  number of mixtures
[D, N] = size(X);
R = zeros(N,K);
product_store = zeros(K,1);
for n = 1:N
    for k = 1:K
        product_store(k,1) = prod(P(k,:)'.^X(:,n).*(1-P(k,:))'.^(1-X(:,n)));
    end
    denominator = pi * product_store;
    for k = 1:K
        numerator = pi(k) * product_store(k,1);
        R(n,k) = numerator / denominator;
    end
end
```

```
function P = calculateParameter(R,X,K)
% Calculate paramter matrix P
 % Input:
 % R: responsiblity matrix
 % X: data
 % K: number of mixtures
 [D N] = size(X);
 P = zeros(K,D);
 for k = 1:K
     P(k,:) =  (R(:,k)'*X'./sum(R(:,k)))';
 end

```

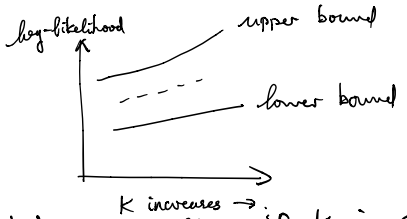
```
function pi = calculateMixingCofficient(R,X,K)
% Calculate Mixing cofficient vector pi
 % Input:
 % R: responsiblity matrix
 % X: data
 % K: number of mixtures
 [D, N] = size(X);
 pi = zeros(1,K);
 for k = 1:K
     pi(k) = sum(R(:,k))/N;
 end

```

```
function log_likelihood = evluateLogLikelihood(R,P,pi,X,K)
% Calculate the updated log likelihood
 % Input:
 % R: responsiblity matrix
 % P: paramter matrix
 % pi: mixing cofficient
 % X: data
 % K: number of mixtures
 [D, N] = size(X);
 log_likelihood = 0;
 product_store = zeros(K,1);
 for n = 1:N
     for k = 1:K
         product_store(k,1) = prod(P(k,:)'.^X(:,n).*(1-P(k,:))'.^(1-X(:,n)));
     end
     log_likelihood = log_likelihood + log(pi * product_store);
 end
```

(e) loglikelihood in bits can be expressed by: $-\frac{\text{log-likelihood}}{\log 2}$. We lose some information here. while gzip compression does not loss any information in compression, it searches for repeated patterns from the data.

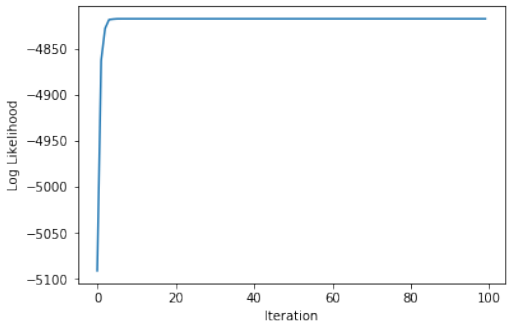
(f) We obtain different log-likelihood results and different parameters with different initial conditions with same K. But the resulting log-likelihood are very close. For example, for K = 4, the log-likelihood converges around -2800 and -3000. If K increases, the log-likelihood increases, and both upper bound and lower bound increases as well. But difference between upper bound and lower bound increases.



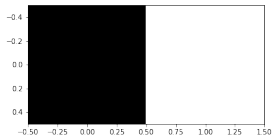
We notice that if K is small, parameter converges to very different values, i.e. it is unstable with small number of clusters. When K increases, EM converges more stable.
Hence the suggestion is to increase K, so that more "features" in the training data could be "explained". (Also likelihood increases).
Also we do multiple initialization to find a "better" local maxima, hopefully the global maxima.

(g). the total cost of encoding in our algorithm is less than that of gzip.

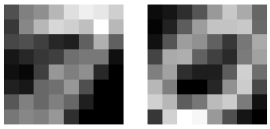
if we increase the value of K, the total cost increases, and it become close to the cost in gzip if K continues to increase.



(a) Log likelihood ($K = 2$)

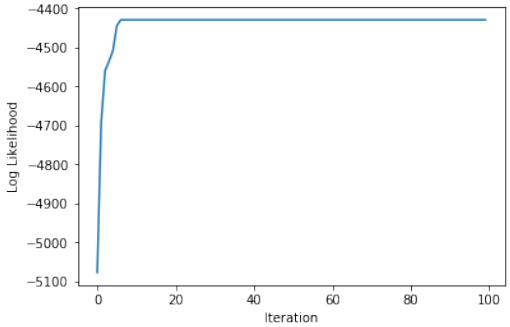


(b) $\widehat{\pi}_{K=2}$

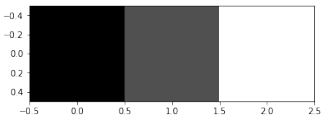


(c) $\widehat{\mathbf{P}}_{K=2}$

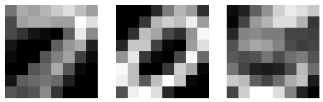
Figure 7: Log Likelihood and estimated parameters when $K = 2$



(a) Log likelihood ($K = 3$)

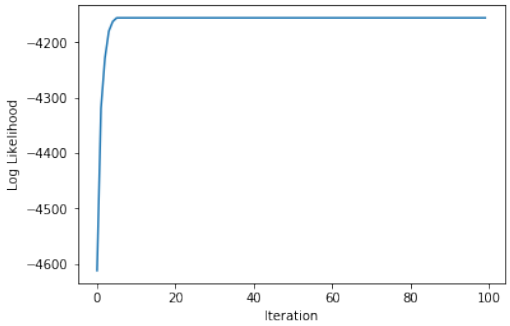


(b) $\widehat{\pi}_{K=3}$

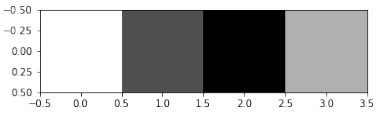


(c) $\widehat{\mathbf{P}}_{K=3}$

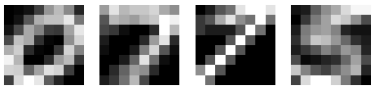
Figure 8: Log Likelihood and estimated parameters when $K = 3$



(a) Log likelihood ($K = 4$)

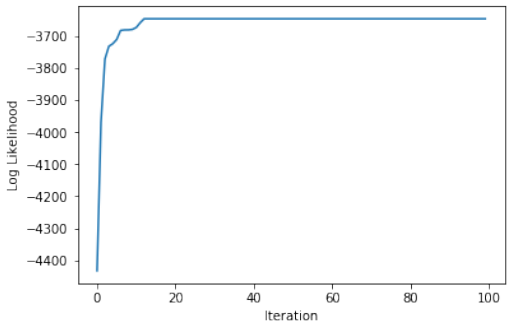


(b) $\widehat{\pi}_{K=4}$

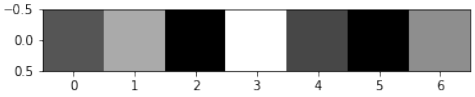


(c) $\widehat{\mathbf{P}}_{K=4}$

Figure 9: Log Likelihood and estimated parameters when $K = 4$



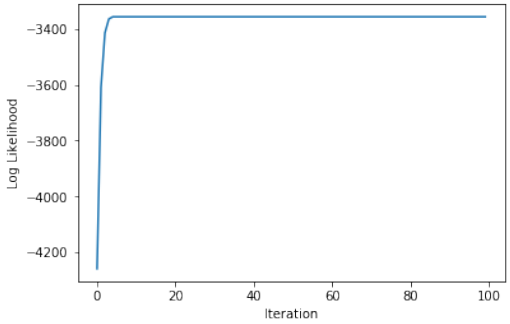
(a) Log likelihood ($K = 7$)



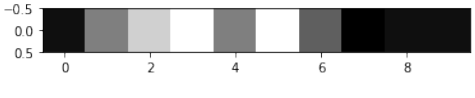
(b) $\widehat{\pi}_{K=7}$

(c) $\widehat{\mathbf{P}}_{K=7}$

Figure 10: Log Likelihood and estimated parameters when $K = 7$



(a) Log likelihood ($K = 10$)



(b) $\widehat{\pi}_{K=10}$

(c) $\widehat{\mathbf{P}}_{K=10}$

Figure 11: Log Likelihood and estimated parameters when $K = 10$

(e)

Here, the length of the naive encoding of these binary data is:

$$8 \times 8 \times 100 = 6400$$

By applying gzip, the 800 bytes large `binarydigits.bin` file can be compressed to 690 bytes (5520 bits). And the log-likelihoods obtained in bits can be calculated as:

$$-\log_2 \text{likelihood}$$

The comparison of these numbers is summarized in Table 2:

According to Table 2, the log-likelihoods in bits for $K = 2, 3, 4, 7, 10$ are all smaller than the length of naive encoding and are also smaller than the gzip size. This fact shows that the results of Mixtures of Multivariate Bernoulli modeling here **perform data compression**.

Since gzip algorithm **performs lossless data compression**, which means that we can reconstruct the original data sets accurately from the compressed one. We can somehow view 5520 bits obtained

K	Log Likelihood in bits	Naive Encoding	Gzip
2	4817		
3	4429		
4	4155	6400	5520
7	3645		
10	3355		

Table 2: Comparison of log-likelihoods in bits, naive encoding, and gzip compression

here as the minimal compressed information we need to carry out the lossless reconstruction of our `binarydigits` data.

However, the log-likelihoods in bits are all **smaller than this number**. Therefore, it can be implied that the Mixtures of Multivariate Bernoulli Model perform a kind of **lossy data compression**, through which we can not reconstruct the original data from the compressed one.

	Mixtures of Multivariate Bernoulli					gzip
	$K = 2$	$K = 3$	$K = 4$	$K = 7$	$K = 10$	
Compression Rate	75.27 %	69.20%	64.92%	56.95%	52.42%	86.25%

Table 3: Compression Rate of Mixtures of Multivariate Bernoulli Model and gzip

By further examining the compression rates shown in Table 3, we know that **the more mixture component** K we set for the Mixtures of Multivariate Bernoulli Model, **the higher compression** we gain, and **the more information we loss** for reconstruction of the original data.

(f)

Here, we run the algorithm for 10 times starting from randomly generated initial conditions. By displaying the estimated $\mathbf{P}$, it can been seen that the estimated clusters indicate different (hand-written) numbers.

The solutions $\left\{\log P(\mathcal{D}|\widehat{\boldsymbol{\pi}}, \widehat{\mathbf{P}}), \widehat{\boldsymbol{\pi}}, \widehat{\mathbf{P}}\right\}$ obtained under different initial conditions are all different. However, if we consider the $\widehat{\boldsymbol{p}}_k$ (row vector of $\mathbf{P}$) to be the same when they indicate the same hand written digits, then we actually gain some same solutions of $\widehat{\mathbf{P}}$ upon permutation within the 10 runs. Also, if we round off $\widehat{\boldsymbol{\pi}}$ to 2 digit decimals, we gain some same solutions of $\widehat{\boldsymbol{\pi}}$ upon permutation within the 10 runs as well.

And we are more likely to obtain the same solution of $\widehat{\mathbf{P}}$ and $\widehat{\boldsymbol{\pi}}$ increases when K is small.

Results of the solutions when $K = 2, 3, 10$ are displayed respectively in Table 4, Table 5 and Figure 12.

Table 4: Cluster Mean and Log Likelihoods obtained in 10 run ($K = 2$)

Run	Cluster Mean	
	digit 7	digit 0
1	0.34	0.66
3	0.29	0.71
7	0.29	0.71
8	0.32	0.68

Run	Cluster Mean	
	digit 5	digit 0
2	0.58	0.42
4	0.58	0.42
5	0.59	0.41
6	0.58	0.42
9	0.6	0.4
10	0.59	0.41

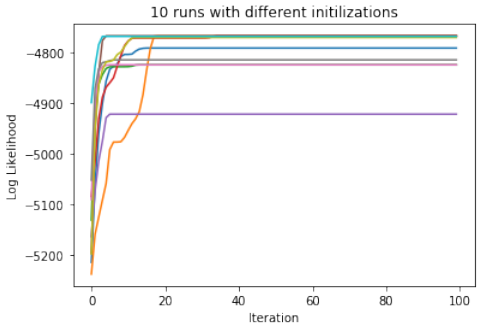
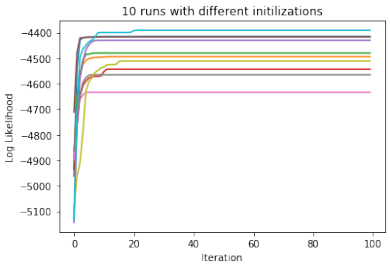
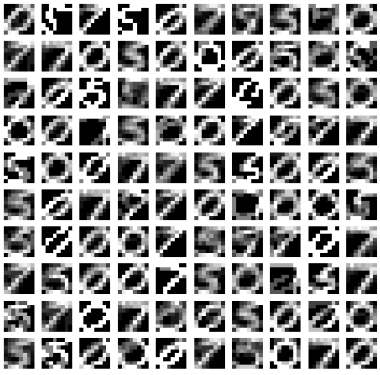


Table 5: Cluster Mean and Log Likelihoods obtained in 10 runs ($K = 3$)

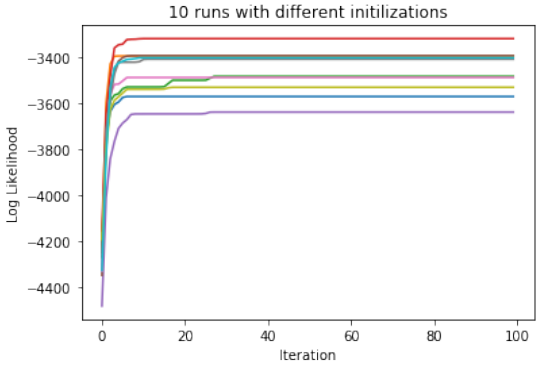
Run	Cluster Mean		
	digit 7	digit 5	digit 0
1	0.15	0.4	0.45
3	0.28	0.23	0.49
4	0.32	0.41	0.27
5	0.32	0.28	0.4
6	0.37	0.33	0.3
9	0.39	0.17	0.44
10	0.25	0.35	0.4

Run	Cluster Mean		
	digit 7	digit 0	digit 0
2	0.49	0.29	0.22
8	0.35	0.45	0.22
Run	Cluster Mean		
	digit 7	digit 7	digit 0
7	0.28	0.11	0.61





(a) $\widehat{\mathbf{P}}$ in 10 runs



(b) Log likelihoods in 10 runs

Figure 12: $\widehat{\mathbf{P}}$ and Log Likelihoods in 10 runs ($K = 10$) under different initialization

We could deduce from the results that the algorithm performs more stable when K is small (most stable when $K = 2$), in the sense that the algorithm finds better clusters and the proportion of the data set is more stable under small K. For instance, when $K = 2$, the algorithm finds clusters of either **digit 7 and 0** pair or **digit 5 and 0** pair, and the proportion is (almost) $3 : 7$ and $6 : 4$

respectively.

However, when K gets larger, the algorithm could not find good clusters. As is shown in Table 5, the algorithm finds clusters of **digit 7, 5 and 0** pair or **digit 7, 7 and 0** pair or **digit 7, 0 and 0** pair, where there is already duplication in digits (7 and 0 appear twice as different clusters). And the cluster mean also becomes very unstable. For even larger K such as $K = 10$, it can been seen from Figure 6 that there are even more duplication of digit clusters (too many digit 7, 5 and 0 !) in each run.

Therefore, we could conclude that the algorithm tends to fall into different local maximum under different initialization, and the larger the K, the more likely solutions will be different. Moreover, the algorithm works well and finds better clusters on relatively small K.

One way to improve the model is to introduce Dirichlet priors over the K multivariate Bernoulli model, including the prior information about what kind of digits we can observe from the data (say, only digit 0, 7 and 5 are observed).

(g)[BONUS]

The total cost of encoding both the model parameters and the data can be computed as

$$\textit{total cost} = -\log_2 \text{likelihood} + K \times D(\text{size of } \mathbf{P}) + K(\text{size of } \pi)$$

and the results are summarized in Table 6:

K	total cost	gzip
2	4947	
3	4624	
4	4415	5520
7	4100	
10	4005	

Table 6: total cost of encoding when $K = 2, 3, 4, 7, 10$ versus gzip

The total cost of encoding when $K = 2, 3, 4, 7, 10$ are still smaller than gzip. However, as the encoding cost of parameters increase when K increase, the total cost does not decrease that much when K gets larger. This might tells us that there exists a trade-off relation between compression of data and complexity of model. When we use complex model (with more parameters), the lossy compression rate might be high but we also have to pay high costs for large parameters size.