Solutions Exercise 3

Question 1

The model is defined as:

$$p(z_n) = \text{Categorical}(\pi) = \pi_k,$$
$$p(x_n \mid z_n = k, \mu) = \prod_{d=1}^{D} \text{Bernoulli}(\mu_{kd}) = \prod_d \mu_{kd}^{x_{nd}} (1 - \mu_{kd})^{1 - x_{nd}}. \tag{1}$$

To simplify the notation, we re-write the discrete variable z_n taking values in $\{1, \ldots, K\}$ as K-dimensional vector that is otherwise zero but takes value 1 for the z_n:th entry. That is, we now have $z_{nk} = 1$ if the nth data point belongs to the kth cluster.

The *observed data log likelihood* for any mixture model with parameters θ (here $\theta = \{\pi, \mu_1, \ldots, \mu_K\}$) is

$$\log \prod_{n=1}^{N} p(x_n \mid \theta) = \sum_n \log \sum_k p(x_n, z_{nk} \mid \theta) = \sum_n \log \sum_k p(z_{nk} \mid \theta) p(x_n \mid z_{nk}, \theta).$$

and plugging in the likelihoods in (1) gives

$$\sum_{n=1}^{N} \log \sum_{k=1}^{K} \left(\pi_k \prod_{d=1}^{D} \mu_{kd}^{x_{nd}} (1 - \mu_{kd})^{1 - x_{nd}} \right). \tag{2}$$

The *complete data log-likelihood* assumes z_{nk} are known and hence we do not need to sum over them. The likelihood for one data point can be written as

$$P(x_n, z_n \mid \theta) = \prod_{k=1}^{K} \left(p(z_{nk} \mid \pi) p(x_n \mid z_{nk}, \mu) \right)^{z_{nk}}.$$

Even though it is a product over k, only a single term remains since $z_{nk} = 1$ only for one k.

The whole complete data log-likelihood is obtained by again plugging in the model (1) and taking logarithm of the above likelihoods:

$$\log P(X, Z \mid \mu, \pi) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \left(\log \pi_k + \sum_{i=1}^{D} x_{ni} \log \mu_{ki} + (1 - x_{ni}) \log(1 - \mu_{ni}) \right). \tag{3}$$

Finally, the *expected complete data log-likelihood* merely takes the expectation of (3) over $p(z_{nk} | \pi_t, \mu_t)$ where π and μ are assumed to be fixed at their current values. It is hence

$$f(\mu, \pi) = \sum_{n=1}^{N} \sum_{k=1}^{K} \mathbb{E}_{p(z_{nk} | \pi_t, \mu_t)}[z_{nk}] \left(\log \pi_k + \sum_{i=1}^{D} x_{ni} \log \mu_{ki} + (1 - x_{ni}) \log(1 - \mu_{ni}) \right). \tag{4}$$

The EM algorithm maximizes the likelihood by alternating two steps. The E-step simply computes the expectation $\mathbb{E}_{p(z_{nk}|\pi_t,\mu_t)}[z_{nk}]$ whereas the M-step maximizes (4) over π and μ.

The E-step is the same for all mixture models and given directly by the Bayes rule:

$$\begin{aligned} p(z_{nk} \mid x_n, \theta) &= \frac{p(x_n, z_{nk} \mid \theta)}{\sum_k p(x_n, z_{nk} \mid \theta)} \\ &= \frac{\pi_k \prod_{d=1}^{D} \mu_{kd}^{x_{nd}} (1-\mu_{kd})^{1-x_{nd}}}{\sum_{k=1}^{K} \pi_k \prod_{d=1}^{D} \mu_{kd}^{x_{nd}} (1-\mu_{kd})^{1-x_{nd}}} \equiv r_{nk}. \end{aligned}$$

The variables r_{nk} are used to store the values of this computation; during M-step r_{nk} is simply some constant.

The M-step can be done independently for each cluster and dimension since the likelihood factorizes over the dimensions. To maximize (4) over some μ_{kd} we analytically solve for $\frac{\partial f(\mu,\pi)}{\partial \mu_{kd}} = 0$. The derivative is

$$\frac{\partial f(\mu, \pi)}{\partial \mu_{kd}} = \sum_{n=1}^{N} r_{nk} \left(\frac{x_{ni}}{\mu_{ki}} - \frac{1 - x_{ni}}{1 - \mu_{ki}} \right)$$

and setting it to zero gives

$$\begin{aligned} \sum_n r_{nk} \frac{x_{nd}}{\mu_{kd}} &= \sum_n r_{nk} \frac{1 - x_{nd}}{1 - \mu_{kd}} \\ (1 - \mu_{kd}) \sum_n r_{nk} x_{nd} &= \mu_{kd} \sum_n r_{nk} (1 - x_{nd}) \\ \mu_{kd} &= \frac{\sum_{n=1}^{N} x_{nd} r_{nk}}{\sum_{n=1}^{N} r_{nk}}. \end{aligned}$$

That is, for every cluster and every dimension we set the parameter μ_{kd} to the ratio of the ones for that dimension, weighted by by r_{nk} indicating how strongly each data point belongs to the cluster.

For optimizing π_k we also need to take into account the constraint $\sum_{k=1}^{K} \pi_k = 1$. We do this using Lagrange multipliers, getting an augmented objective $Q(\pi, \mu, \lambda) = f(\pi, \mu) - \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right)$. By differentiating that wrt to π_k and setting the derivative to zero we get

$$\frac{\sum_{n=1}^{N} r_{nk}}{\pi_k} - \lambda = 0 \rightarrow \pi_k = \frac{\sum_{n=1}^{N} r_{nk}}{\lambda}.$$

Plugging in the constraint $\sum_k \pi_k = 1$ gives

$$\sum_k \pi_k = \sum_k \frac{\sum_{n=1}^{N} r_{nk}}{\lambda} = 1,$$

from which we can solve $\lambda = \sum_k \sum_n r_{nk}$. This gives the final update

$$\pi_k = \frac{\sum_{n=1}^{N} r_{nk}}{\sum_{k=1}^{K} \sum_{n=1}^{N} r_{nk}}.$$

This step is the same for all mixture models since the term $p(x_n \mid z_n, \mu)$ does not depend on π, so in practice this derivation need not be repeated for new models. Intuitively, the result simply gives the ratio of the weights r_{nk} falling into each cluster.

Question 2

This exercise studied the spectral clustering algorithm consisting of three steps: Find a graph representation for the data, compute the eigen-decomposition of its graph Laplacian, and cluster the data based on the eigenvectors corresponding to the smallest eigenvalues. The first and last step of this algorithm have a few parameters that dramatically alter the performance of the model, and hence the point of this exercise was to notice that the procedure is somewhat fragile and needs to be implemented carefully for real clustering tasks. The full code is available in Moodle.

a) An example code for creating the graph is given below; with flag=1 it connects to nodes within a given distance and otherwise it connects to the K nearest neighbors:

```
def spectral(X, flag, e, A, M):
    """
    X: data
    flag: to pick distance (e) or closest neighbours(A) for adjacency matrix calculations
    M: dimension of the transformed data
    Return: transformed data, eigenvalues, eigenvectors, order index(ascending order)
    """
    N = X.shape[0]
    # calculate the Euclidean distances
    dist_vec = [np.linalg.norm(X[i,:]-X[j,:]) for i in range(N) for j in range(N)]
    # reshape the distances into a matrix of N x N dimensionality
    dist_mat = np.reshape(dist_vec,(N,N))
    # flag option lets you choose between types of adjacency matrices construction
    # if flag == 1; we use the distance based method
    # if flag == 2 we use A closest neighbors construction
    if flag == 1:
        W = 1*(dist_mat <= e)
    elif flag == 2:
        W = np.zeros((N,N))
        for i in range(N):
            tempvec1 = [(dist_mat[i,jj],jj) for jj in range(N)]
            tempvec1.sort()
            dsorder1 = [tempvec1[ii][1] for ii in range(N)]
            pick = dsorder1[1:(1+A)]
            W[i,pick] = 1
            W[pick,i] = 1
    else:
        print("Error: Use flag == 1 or 2")
```

```
    return None
# Diagonal matrix (D)
D = np.eye(N) *  np.sum(W,1)
# Graph Laplacian (L)
L = D - W
# get the eigenvalues and eigenvectors of L
evals, evecs = np.linalg.eig(L)
tempvec = [(evals[i],i) for i in range(N)]
# sort the eigenvalues in ascending order
tempvec.sort()
dsorder = [tempvec[i][1] for i in range(len(evecs))]
# Transformed data based on M
temp_Y = np.zeros((N,M))
for jj in range(M):
    temp_Y[:,jj] = evecs[:,dsorder[jj]]
return [temp_Y, evals, evecs, dsorder]
```

b) The eigenvectors from eigendecomposition of the graph Laplacian is illustrated in Figure 1. The first eigenvector is constant because the graph is fully connected, but the following eigenvectors show some relationship between the assumed cluster structure. If we managed to split the graph into two disjoint subgraphs corresponding to the clusters we would get two eigenvectors that directly reveal the clusters, but since this is not the case we see only some perturbed versions of the ideal case.

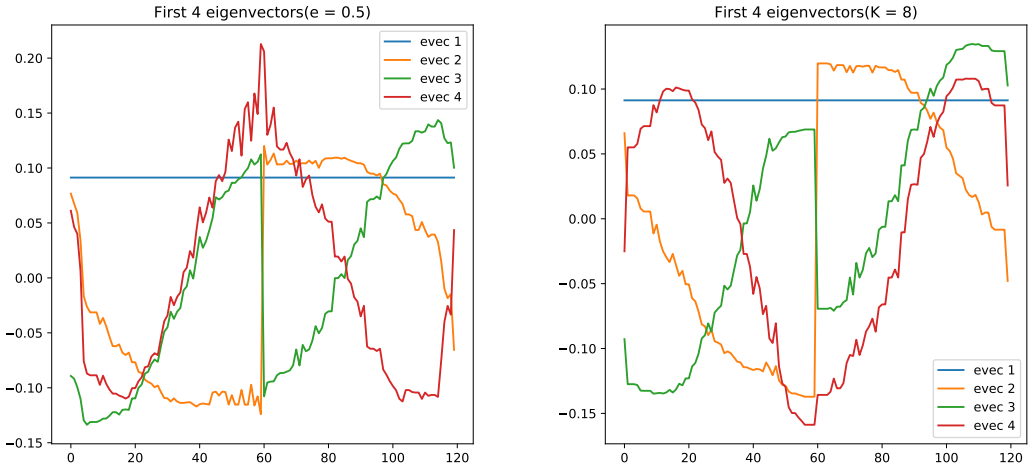

Figure 1: For case (e = 0.5): The first eigenvector is constant, indicating that the graph is connected, but the 2nd and 3rd clearly reveal the cluster structure. Similarly, case (k = 8): The result is very similar to case (e = 0.5), demonstrats that the same core idea works even if the graph representation is different.

c) The scatter-plot (Figure 2) clearly shows how the first dimension is not very helpful, but especially the 2nd and 3rd dimension are clearly informative of the cluster structure.

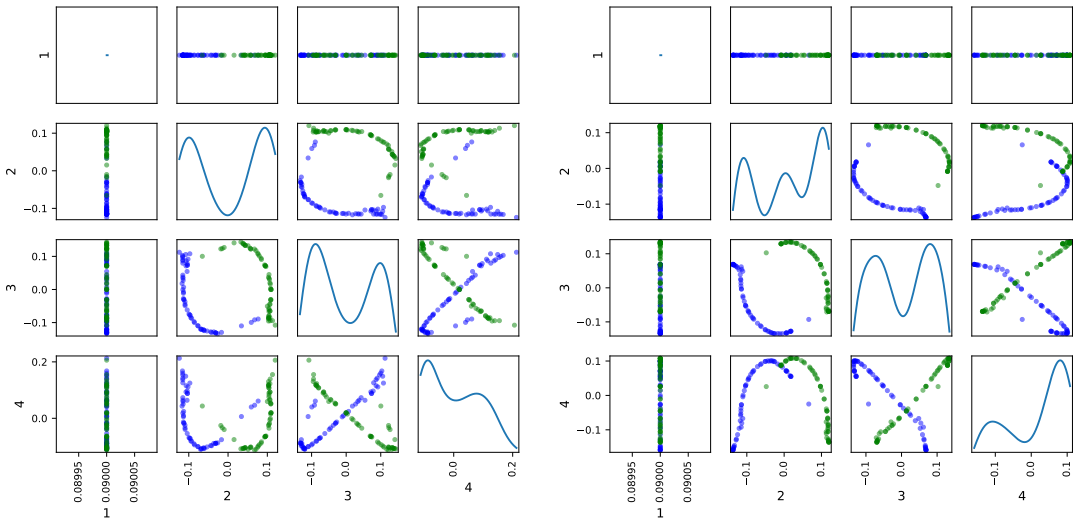

Figure 2: The figure shows the scatter-plot of first four eigenvectors for cases $e = 0.5$ (left) and $A = 8$ (right). The scatter-plot of the first two dimensions of the new representation matrix results in a line, and clusters are slightly noticable but when we use scatter plot involving second, third and fourth dimensions we see the clusters.

d) The clustering results are shown in Figure 3, indicating that K-means in the new representation reveals the intuitive clusters much better than K-means in the original data space.

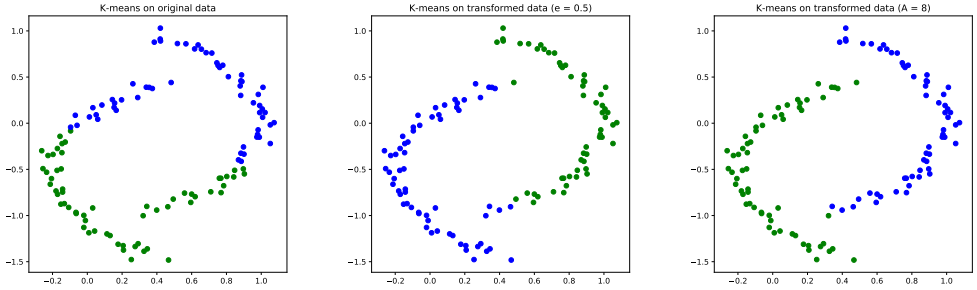

Figure 3: K-means in the original data space (left) does not reveal the natural clusters since it assumes each cluster represents a sphere. Both spectral clustering variants (middle for $e = 0.5$ and right for $A = 8$) find almost the correct solution.

The key finding regarding the parameters e and A is that with too small values the graph separates into disconnected components, and the graph Laplacian then has

multiple eigenvalues of zero. The corresponding eigenvectors capture the disconnected subgraphs well, but typically this is not what we want – they do not correspond to the natural clusters but instead each subset of nodes that happened to be separated from the rest when constructing the graph becomes one cluster. Often these consists of just 1-2 samples. With too large e and k the graph no longer represents the underlying data but instead all nodes are connected to most nodes, and the algorithm does not do anything reasonable.

M controls how many dimensions we keep, and it is important to get this right. With too small value there is not enough information to separate the clusters (especially with $M = 1$ all data points are represented with the same value). On the other hand, too large M starts to decrease the clustering performance because the eigenvectors corresponding to larger eigenvalues are not related to the cluster structure at all. The idea of spectral clustering is to look at the eigenvectors corresponding to eigenvalues that are close to zero, whereas from the perspective of the clustering the ones corresponding to large eigenvalues are pure noise. Adding noise dimensions certainly drops the K-means accuracy.

Question 3

The fact that the outputs of this procedure produce orthonormal matrices can be shown by induction.

For the first component the first operation does nothing and the second operation normalizes the vector since $\mathbf{w}^T\mathbf{w} = \frac{\mathbf{v}^T}{\sqrt{\mathbf{v}^T\mathbf{v}}}\frac{\mathbf{v}}{\sqrt{\mathbf{v}^T\mathbf{v}}} = \frac{\mathbf{v}^T\mathbf{v}}{\mathbf{v}^T\mathbf{v}} = 1$. Hence, it clearly is a unit vector.

Next assume we have already found k orthonormal columns and prove that the algorithm guarantees the $(k+1)$th column satisfies $\mathbf{w}_{k+1}^T\mathbf{w}_{k+1} = 1$ and $\mathbf{w}_{k+1}^T\mathbf{w}_l = 0$ for all $l \leq k$. The former follows directly from the second normalization analogous to the first vector. What remains to show is the orthogonality w.r.t to the previous k vectors, denoted by $\mathbf{w}_l$. Using $i = k+1$ and plugging the update rule into the definition of orthogonality gives

$$\begin{aligned}\mathbf{w}_l^T\mathbf{v}_i &= \mathbf{w}_l^T(\mathbf{w}_i - \sum_{j\leq k}(\mathbf{w}_i^T\mathbf{w}_j)\mathbf{w}_j) \\ &= \mathbf{w}_l^T\mathbf{w}_i - \sum_{j\leq k}(\mathbf{w}_i^T\mathbf{w}_j)(\mathbf{w}_l^T\mathbf{w}_j)\end{aligned}$$

Since $\mathbf{w}_l^T\mathbf{w}_j = 0$ for $l \neq j$ and $\mathbf{w}_l^T\mathbf{w}_l = 1$ for any l, the only non-zero term in the summation is $\mathbf{w}_i^T\mathbf{w}_l$. Hence, $\mathbf{w}_l^T\mathbf{v}_i = \mathbf{w}_l^T\mathbf{w}_i - \mathbf{w}_i^T\mathbf{w}_l = 0$. This concludes the proof.