40.319 Statistical and Machine Learning
Spring 2020 Homework 3 Solutions

Problem 1 (10 points)

In this problem, we will implement the EM algorithm for clustering. Start by importing the required packages and preparing the dataset.

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

from numpy import linalg as LA
from matplotlib.patches import Ellipse
from sklearn.datasets.samples_generator import make_blobs
from scipy.stats import multivariate_normal

K = 3
NUM_DATAPTS = 150

X, y = make_blobs(n_samples=NUM_DATAPTS, centers=K, shuffle=False,
                                    random_state=0, cluster_std=0.6)
g1 = np.asarray([[2.0, 0], [-0.9, 1]])
g2 = np.asarray([[1.4, 0], [0.5, 0.7]])
mean1 = np.mean(X[:int(NUM_DATAPTS/K)])
mean2 = np.mean(X[int(NUM_DATAPTS/K):2*int(NUM_DATAPTS/K)])
X[:int(NUM_DATAPTS/K)] = np.einsum('nj,ij->ni',
                X[:int(NUM_DATAPTS/K)] - mean1, g1) + mean1
X[int(NUM_DATAPTS/K):2*int(NUM_DATAPTS/K)] = np.einsum('nj,ij->ni',
                X[int(NUM_DATAPTS/K):2*int(NUM_DATAPTS/K)] - mean2, g2) + mean2
X[:,1] -= 4
```

(a) Randomly initialize a numpy array *mu* of shape (K, 2) to represent the mean of the clusters, and initialize an array *cov* of shape (K, 2, 2) such that *cov*[k] is the identity matrix for each k. *cov* will be used to represent the covariance matrices of the clusters. Finally, set π to be the uniform distribution at the start of the program.

(b) Write a function to perform the E-step:

```
def E_step():
        gamma = np.zeros((NUM_DATAPTS, K))
        ...
        ...
        return gamma
```

(c) Write a function to perform the M-step:

```
def M_step(gamma):
        ...
        ...
```

(d) Now write a loop that iterates through the E and M steps, and terminates after the change in log-likelihood is below some threshold. At each iteration, print out the log-likelihood, and use the following function to plot the progress of the algorithm:

```
def plot_result(gamma=None):
        ax = plt.subplot(111, aspect='equal')
        ax.set_xlim([-5, 5])
        ax.set_ylim([-5, 5])
        ax.scatter(X[:, 0], X[:, 1], c=gamma, s=50, cmap=None)

        for k in range(K):
            l, v = LA.eig(cov[k])
            theta = np.arctan(v[1,0]/v[0,0])

            e = Ellipse((mu[k, 0], mu[k, 1]), 6*l[0], 6*l[1],
                    theta * 180 / np.pi)
            e.set_alpha(0.5)
            ax.add_artist(e)

        plt.show()
```

(e) Use sklearn's KMeans module

https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

to perform K-means clustering on the dataset, and compare both clustering results.

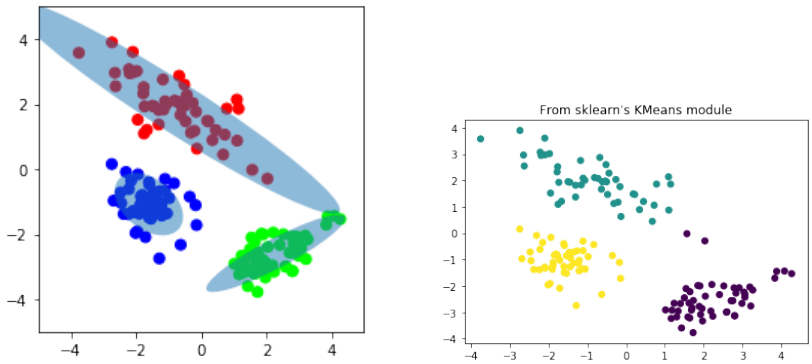
Solution.

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

from numpy import linalg as LA
from matplotlib.patches import Ellipse
from sklearn.datasets.samples_generator import make_blobs
from scipy.stats import multivariate_normal

K = 3
NUM_DATAPTS = 150

X, y = make_blobs(n_samples=NUM_DATAPTS, centers=K, shuffle=False,
                                        random_state=0, cluster_std=0.6)
g1 = np.asarray([[2.0, 0], [-0.9, 1]])
g2 = np.asarray([[1.4, 0], [0.5, 0.7]])
mean1 = np.mean(X[:int(NUM_DATAPTS/K)])
mean2 = np.mean(X[int(NUM_DATAPTS/K):2*int(NUM_DATAPTS/K)])
X[:int(NUM_DATAPTS/K)] = np.einsum('nj,ij->ni',
                X[:int(NUM_DATAPTS/K)] - mean1, g1) + mean1
X[int(NUM_DATAPTS/K):2*int(NUM_DATAPTS/K)] = np.einsum('nj,ij->ni',
                X[int(NUM_DATAPTS/K):2*int(NUM_DATAPTS/K)] - mean2, g2) + mean2
X[:,1] -= 4

#For 1(a)
pi = np.ones(K)
```

```
pi /= np.sum(pi)
mu = 10 * np.random.rand(K, 2) - 5
cov = np.zeros((K, 2, 2))
for i in range(K):
        cov[i] = np.eye(2)

#For 1(b)
def E_step():
        gamma = np.zeros((NUM_DATAPTS, K))
        for k in range(K):
                numer = pi[k] * multivariate_normal.pdf(X, mu[k], cov[k])
                denom = 0
                for j in range(K):
                        denom +=  pi[j] * multivariate_normal.pdf(X, mu[j], cov[j])

                gamma[:,k] = numer / denom
        return gamma

#For 1(c)
def M_step(gamma):
        for k in range(K):
                pi[k] = np.mean(gamma[:, k])

                denom = np.sum(gamma[:,k])
                mu[k,0] = np.sum(X[:,0] * gamma[:, k]) / denom
                mu[k,1] = np.sum(X[:,1] * gamma[:, k]) / denom

                sample_cov = np.reshape(np.einsum('ni,nj->nij', X - mu[k], X - mu[k]),
                                (NUM_DATAPTS, 4))
                cov[k] = np.reshape(np.sum(sample_cov * np.reshape(gamma[:, k],
                                (NUM_DATAPTS, 1)), axis=0) / denom, (2, 2))

#For 1(d)
def main():
        fig = plt.figure()
        plot_result()

        t = 0
        prev_ll = -5000
        cur_ll = -4000
        while abs(prev_ll - cur_ll) > 0.1:
                prev_ll = cur_ll
                t += 1
                print("Time: ", t)

                gamma = E_step()
                M_step(gamma)

                cur_ll = 0
                for i in range(NUM_DATAPTS):
                        likelihood = 0
                        for k in range(K):
```

```
                    likelihood += pi[k] * multivariate_normal.pdf(X[i,:],
                                                mu[k], cov[k])
            cur_ll += np.log(likelihood)
        print("Log-likelihood: ", cur_ll)

        plot_result(gamma)

#For 1(e)
from sklearn.cluster import KMeans

y_pred = KMeans(n_clusters=K).fit_predict(X)

plt.figure()
plt.scatter(X[:, 0], X[:, 1], c=y_pred)
plt.title("From sklearn's KMeans module")
plt.show()
```

Comparison: Both clustering results look similar with slight differences (2 data points).



(a) EM algorithm (b) KMeans algorithm

Figure 1: Comparison between both clustering results

■

Problem 2 (5 points)

Let p and q be distributions on $\{1, 2, 3, 4, 5\}$ such that $p_1 = \frac{1}{8}$, $p_2 = \frac{1}{2}$, $p_3 = p_4 = p_5 = \frac{1}{8}$, and $q_1 = \frac{1}{4}$, $q_2 = q_3 = \frac{1}{8}$, $q_4 = q_5 = \frac{1}{4}$.

(a) Compute the cross-entropy $H(p, q)$ in bits. Is $H(q, p) = H(p, q)$?

(b) Compute the entropies $H(p)$ and $H(q)$ in bits.

(c) Compute the KL-divergence $D_{KL}(p \,|\, q)$ in bits.

Show all working and leave your answers in fractions.

Solution.

(a) We have

$$\begin{aligned} H(p,q) &= \frac{1}{8}\log_2 4 + \frac{1}{2}\log_2 8 + \frac{1}{8}\log_2 8 + \frac{1}{8}\log_2 4 + \frac{1}{8}\log_2 4 \\ &= \frac{2}{8} + \frac{3}{2} + \frac{3}{8} + \frac{2}{8} + \frac{2}{8} = \frac{21}{8}. \end{aligned}$$

On the other hand,

$$\begin{aligned} H(q,p) &= \frac{1}{4}\log_2 8 + \frac{1}{8}\log_2 2 + \frac{1}{8}\log_2 8 + \frac{1}{4}\log_2 8 + \frac{1}{4}\log_2 8 \\ &= \frac{3}{4} + \frac{1}{8} + \frac{3}{8} + \frac{3}{4} + \frac{3}{4} = \frac{22}{8} \neq H(p,q). \end{aligned}$$

(b) We have

$$\begin{aligned} H(p) &= \frac{1}{8}\log_2 8 + \frac{1}{2}\log_2 2 + \frac{1}{8}\log_2 8 + \frac{1}{8}\log_2 8 + \frac{1}{8}\log_2 8 \\ &= \frac{3}{8} + \frac{1}{2} + \frac{3}{8} + \frac{3}{8} + \frac{3}{8} = 2 \end{aligned}$$

and

$$\begin{aligned} H(q) &= \frac{1}{4}\log_2 4 + \frac{1}{8}\log_2 8 + \frac{1}{8}\log_2 8 + \frac{1}{4}\log_2 4 + \frac{1}{4}\log_2 4 \\ &= \frac{2}{4} + \frac{3}{8} + \frac{3}{8} + \frac{2}{4} + \frac{2}{4} = \frac{9}{4}. \end{aligned}$$

(c) $D_{KL}(p\,|\,q) = \frac{21}{8} - 2 = \frac{5}{8}$.

■

Problem 3 (10 pts)

(a) Perform singular value decomposition (SVD) on the following matrix

$$X = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = U\Sigma V^T.$$

(b) For a general design matrix X, why are the columns of the transformed matrix $T = XV$ orthogonal?

Solution. (a) We compute X^TX and get

$$X^TX = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}.$$

The eigenvalues, in order of decreasing magnitude, are $\lambda_1 = 3$ and $\lambda_2 = 1$, and the corresponding singular values are $\sigma_1 = \sqrt{3}, \sigma_2 = 1$. This means that

$$\Sigma = \begin{bmatrix} \sqrt{3} & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}.$$

The corresponding orthonormal eigenvector of λ_1 is $v_1 = \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]^T$, and that of λ_2 is $v_2 = \left[\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}\right]^T$. Thus

$$V = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix}.$$

To obtain U, we compute

$$XX^T = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix},$$

which we know will have eigenvalues 3, 1 and 0. The corresponding orthonormal eigenvectors are $u_1 = \frac{1}{\sqrt{6}}[2, 1, 1]^T$, $u_2 = \frac{1}{\sqrt{2}}[0, 1, -1]^T$ and $u_3 = \frac{1}{\sqrt{3}}[-1, 1, 1]^T$ respectively. Hence

$$X = \begin{bmatrix} \frac{2}{\sqrt{6}} & 0 & \frac{-1}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} \sqrt{3} & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{bmatrix}^T.$$

(b) We have

$$T^T T = V^T X^T X V = V^T (V \Sigma^T U^T)(U \Sigma V^T) V = \Sigma^T \Sigma,$$

which is a diagonal matrix. This means that $\langle t_i, t_j \rangle = 0$ for all $i \neq j$ as this quantity is equal to the ij^{th} entry of the matrix.

■

Problem 4 (4 points)

In this problem, we will perform principal component analysis (PCA) on sklearn's diabetes dataset. Start by importing the required packages and load the dataset.

```
import numpy as np
from sklearn import decomposition
from sklearn import datasets

X = datasets.load_diabetes().data
```

You can find out more on how to use sklearn's PCA module from:
https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

For this problem, make sure the design matrix is first normalized to have zero mean and unit standard deviation for each column.

(a) Write code to print the matrix V that will be used to transform the dataset, and print all the singular values.

(b) Now perform PCA on the dataset and print out the 3 most important components for the first 10 data-points.

As this problem is short, print out your script and results, and include it with your hardcopy submission. You do not have to upload the solution.

Solution.
```
import numpy as np
from sklearn import decomposition
from sklearn import datasets

X = datasets.load_diabetes().data

#For 4(a)
X = preprocessing.scale(X)
pca = decomposition.PCA(n_components=10)
pca.fit(X)

V = pca.transform(np.eye(10))
print("Transformation matrix V: ", V)
print("Singular values: ", pca.singular_values_)

#For 4(b)
X = pca.transform(X)
print(X[:10, :3])
```

■

Problem 5 (5 points)

An AR(2) model assumes the form

$$r_t = \phi_0 + \phi_1 r_{t-1} + \phi_2 r_{t-2} + a_t,$$

where a_t is a white noise sequence. Show that if the model is stationary, then

(a) $E(r_t) = \frac{\phi_0}{1-\phi_1-\phi_2}$ (assume $\phi_1 + \phi_2 \neq 1$);

(b) the ACF is given by

$$\rho(1) = \frac{\phi_1}{1-\phi_2}, \qquad \rho(s) = \phi_1\rho(s-1) + \phi_2\rho(s-2), \quad \forall\, s \geq 2.$$

Solution. (a) Applying expectation on both sides, we have

$$\mathbb{E}[r_t] = \mu = \phi_0 + \phi_1\mu + \phi_2\mu,$$

and thus $\mu = \frac{\phi_0}{1-\phi_1-\phi_2}$.

(b) Using $\phi_0 = \mu(1-\phi_1-\phi_2)$, we can rewrite the AR(2) equation as

$$r_t - \mu = \phi_1(r_{t-1}-\mu) + \phi_2(r_{t-2}-\mu) + a_t.$$

Now multiplying both sides by $r_{t-s} - \mu$ and taking expectations, and using the fact that $\mathrm{Cov}(r_{t-s}, a_t) = 0$, we get

$$\mathrm{Cov}(r_t, r_{t-s}) = \phi_1\mathrm{Cov}(r_{t-1}, r_{t-s}) + \phi_2\mathrm{Cov}(r_{t-2}, r_{t-s}).$$

Dividing everything by $\mathrm{Var}(r_t)$, we obtain

$$\rho(s) = \phi_1 \rho(s-1) + \phi_2 \rho(s-2), \quad \forall\, s \geq 2,$$

and

$$\rho(1) = \phi_1 + \phi_2 \rho(-1) = \phi_1 + \phi_2 \rho(1) \implies \rho(1) = \frac{\phi_1}{1-\phi_2}.$$

■