Saad Mufti

CS 539 - HW 3

Conceptual and Theoretical Questions

Q1

It is indeed true that a and b are conditionally dependent, as the marginal distribution of a is

$$P(a = 0) = 0.192 + 0.144 + 0.048 + 0.216 = 0.6$$
$$P(a = 1) = 0.192 + 0.064 + 0.048 + 0.096 = 0.4$$

and the marginal distribution of b is

$$P(b = 0) = 0.192 + 0.144 + 0.192 + 0.064 = 0.592$$
$$P(b = 1) = 0.048 + 0.216 + 0.048 + 0.096 = 0.408$$

a and b can be proved marginally dependent by providing a counterexample to fulfill $p(a, b) \neq p(a)p(b)$, so take $a = 1$, and $b = 0$:

$$p(a, b) = 0.192 + 0.064 = 0.256$$
$$p(a)p(b) = 0.4 \cdot 0.592 = 0.2368$$

Thus, a and b are marginally dependent as $p(a, b) \neq p(a)p(b)$.

To verify a and b are conditionally independent on c, show $p(a, b|c) = p(a|c)p(b|c)$:

Conditional probability can be found with

$$p(x|y) = \frac{p(x \cap y)}{p(y)}$$

, so

$$p(c = 0) = 0.192 + 0.048 + 0.192 + 0.048 = 0.48$$
$$p(c = 1) = 0.144 + 0.216 + 0.064 + 0.096 = 0.52$$
$$P(a = 0|c = 0) = \frac{0.048 + 0.192}{0.48} = \frac{0.24}{0.48} = 0.5$$
$$P(a = 0|c = 1) = \frac{0.144 + 0.216}{0.52} = \frac{0.36}{0.52} = 0.6923$$
$$P(a = 1|c = 0) = \frac{0.192 + 0.48}{0.48} = \frac{0.24}{0.48} = 0.5$$
$$P(a = 1|c = 1) = \frac{0.064 + 0.096}{0.52} = \frac{0.16}{0.52} = 0.3077$$
$$P(b = 0|c = 0) = \frac{0.192 + 0.192}{0.48} = \frac{0.384}{0.48} = 0.8$$
$$P(b = 0|c = 1) = \frac{0.144 + 0.064}{0.52} = \frac{0.208}{0.52} = 0.4$$
$$P(b = 1|c = 0) = \frac{0.048 + 0.048}{0.48} = \frac{0.096}{0.48} = 0.2$$
$$P(b = 1|c = 1) = \frac{0.216 + 0.096}{0.52} = \frac{0.312}{0.52} = 0.6$$

For $c = 0$:

$$P(a = 0, b = 0|c = 0) = \frac{0.192}{0.48} = p(a = 0|c = 0) \cdot p(b = 0|c = 0) = 0.5 \cdot 0.8 = 0.4$$
$$P(a = 0, b = 1|c = 0) = \frac{0.048}{0.48} = p(a = 0|c = 0) \cdot p(b = 1|c = 0) = 0.5 \cdot 0.2 = 0.1$$
$$P(a = 1, b = 0|c = 0) = \frac{0.192}{0.48} = p(a = 0|c = 0) \cdot p(b = 0|c = 0) = 0.5 \cdot 0.8 = 0.4$$

$$P(a=1, b=1|c=0) = \frac{0.048}{0.48} = p(a=1|c=0) \cdot p(b=1|c=0) = 0.5 \cdot 0.2 = 0.1$$

For $c = 1$:

$$P(a=0, b=0|c=1) = \frac{0.144}{0.52} = P(a=0|c=1) \cdot P(b=0|c=1) = 0.6923 \cdot 0.4 = 0.2769$$
$$P(a=0, b=1|c=1) = \frac{0.216}{0.52} = P(a=0|c=1) \cdot P(b=1|c=1) = 0.6923 \cdot 0.6 = 0.4154$$
$$P(a=1, b=0|c=1) = \frac{0.064}{0.52} = P(a=1|c=1) \cdot P(b=0|c=1) = 0.3077 \cdot 0.4 = 0.1231$$
$$P(a=1, b=1|c=1) = \frac{0.096}{0.52} = P(a=1|c=1) \cdot P(b=1|c=1) = 0.3077 \cdot 0.6 = 0.1846$$

Thus, a and b are conditionally independent on c.

Q2

Distribution of $p(a)$:

$$P(a=0) = 0.192 + 0.144 + 0.048 + 0.216 = 0.6$$
$$P(a=1) = 0.192 + 0.064 + 0.048 + 0.096 = 0.4$$

Distribution of $p(b|c)$:

$$P(b=0|c=0) = \frac{0.192 + 0.192}{0.48} = \frac{0.384}{0.48} = 0.8$$
$$P(b=0|c=1) = \frac{0.144 + 0.064}{0.52} = \frac{0.208}{0.52} = 0.4$$
$$P(b=1|c=0) = \frac{0.048 + 0.048}{0.48} = \frac{0.096}{0.48} = 0.2$$
$$P(b=1|c=1) = \frac{0.216 + 0.096}{0.52} = \frac{0.312}{0.52} = 0.6$$

Distribution of $p(c|a)$:

$$P(c=0|a=0) = \frac{0.192 + 0.048}{0.6} = \frac{0.24}{0.6} = 0.4$$
$$P(c=0|a=1) = \frac{0.192 + 0.048}{0.4} = \frac{0.24}{0.4} = 0.6$$
$$P(c=1|a=0) = \frac{0.144 + 0.216}{0.6} = \frac{0.36}{0.6} = 0.6$$
$$P(c=1|a=1) = \frac{0.064 + 0.096}{0.4} = \frac{0.16}{0.4} = 0.4$$

The following table ensures $p(a,b,c) = p(a)p(c|a)p(b|c)$:

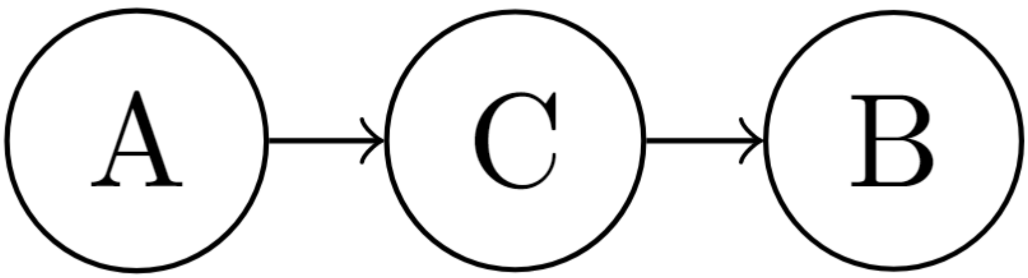
a	b	c	$p(a,b,c)$	**$p(a)p(c**	**a)p(b**	**c)$**
0	0	0	0.192	$0.6 \cdot 0.4$ $\cdot\, 0.8$ $= 0.192$		
0	0	1	0.144	$0.6 \cdot 0.6$ $\cdot\, 0.4$ $= 0.144$		
0	1	0	0.048	$0.6 \cdot 0.4$ $\cdot\, 0.2$ $= 0.048$		
0	1	1	0.216	$0.6 \cdot 0.6$ $\cdot\, 0.6$ $= 0.216$		
1	0	0	0.192	$0.4 \cdot 0.6$ $\cdot\, 0.8$ $= 0.192$		
1	0	1	0.064	$0.4 \cdot 0.4$ $\cdot\, 0.4$ $= 0.064$		

a	b	c	$p(a,b,c)$	**$p(a)p(c**	**a)p(b**	**c)$**
1	1	0	0.048	$0.4 \cdot 0.6 \cdot 0.2 = 0.048$		
1	1	1	0.096	$0.4 \cdot 0.4 \cdot 0.6 = 0.096$		

Following the general form

$$p(x|pa(x))p(pa(x))$$

The directed graph would look like the following



since the given expression indicates a is the top parent of the graph, while c is a direct descendant of b and c is a direct descendant of a

Q3

In the joint distribution table of x and y, there should only be non-zero values in the column/row for $x = \hat{x}$ and $y = \hat{y}$, whose values are arbitrary. The intersection of the x and y row/columns should be 0 since the condition here is $p(\hat{x}, \hat{y}) = 0$. From here, the construction of the joint distribution comes down to choosing values for $\hat{x}$ and $\hat{y}$

$x \ y$	0	1	2	$p(x)$
0	0	0.3	0	0.3
1	0.3	0	0.2	0.5
2	0	0.2	0	0.2
$p(y)$	0.3	0.5	0.2	1

Evidently, $\hat{x} = 1$ and $\hat{y} = 1$ (although there is no reason for choosing these values. This could have easily been done for any other $\hat{x}$ and $\hat{y}$). The marginal distribution $p(\hat{x})$ and $p(\hat{y})$ are maximized at 0.5 each, shown in the margins of the table. Lastly, $p(x = 0, y = 0) = 0$, so the above table meets all the conditions.

Q4

$p(\theta|x) \propto p(x|\theta)p(\theta)$ Bayes' Theorem

$\ln(p(\theta|x)) \propto \ln p(x|\theta) + \ln p(\theta)$ Log Likelihood

$\propto \ln\left(\sum_z p(x, z|\theta)\right) \ln p(\theta)$ (9.29, Bishop)

$= \ln p(\theta) \sum_z p(x, z|\theta)$

$Q(\theta, \theta^{old}) = \sum_z p(z|x, \theta^{old}) \ln p(x, z|\theta)$ (9.30, Bishop)

$Q'(\theta, \theta^{old}) = \sum_z p(z|x, \theta^{old}) \ln[p(x, z|\theta) p(\theta)]$

$\ln p(x, z|\theta) \sum_z p(z|x, \theta^{old}) + \ln p(\theta) \sum_z p(z|x, \theta^{old})$

$\downarrow \qquad\qquad = 1$

$Q' = Q(\theta, \theta^{old}) + \ln p(\theta)$

Likelihood

$p(t|X, w, \beta) = \prod N(t_n | w^T\phi(x_n), \beta^{-1})$

$p(w, \beta) = N(w|m_0, \beta^{-1} S_0)\, Gam(\beta|a_0, b_0)$

Prior ↑

Prior $\quad p(w, \beta) = \frac{\beta}{S_0\sqrt{2\pi}} \exp\left(\frac{\beta^2}{2S_0^2}(w - m_0)^2\right) \cdot \frac{b_0^{a_0}}{\Gamma(a)} \beta^{a_0-1} e^{-b_0\beta}$

$\propto \frac{\beta}{S_0} \exp\left(\frac{\beta^2}{2S_0^2}(w - m_0)^2\right) b_0^{a_0} \beta^{a_0-1} e^{-b_0\beta} \quad \propto \exp\left((w - m_0)^T \frac{\beta^2}{S_0^2}(w - m_0)\right) b_0^{a_0} \beta^{a_0-1} e^{-b_0\beta}$

Likelihood $\quad p(t|X, w, \beta) = \frac{\beta^{1/2}}{\sqrt{2\pi}} \exp\left(-\frac{\beta^2}{2}(t_n - w^T\phi(x_n))^2\right)$

$\propto \beta^{N/2} \exp\left(-\frac{\beta^2}{2}(t_N - w^T\phi(x_n))^2\right)$

$\propto \beta^{N/2} \exp\left(-\frac{\beta^2}{2}(t_N - w^T\phi(x_n))^T (t_N - w^T\phi(x_N))\right)$

Mult likelihood, prior

$\exp\left(-(w - m_0)^T \frac{\beta^2}{2S_0^2}(w - m_0)\right) \exp\left(-\frac{\beta}{2}(t_N - w^T\phi(x_n))^T (t_N - w^T\phi(x_N))\right) b_0^{a_0} \beta^{a_0-1} e^{-b_0\beta}$

$\lambda = \left(\frac{1}{\sigma^2}\right)^2$

$\propto \exp\left(-\frac{1}{2} w^T \frac{\beta}{S_0} w + \beta m_0^T S_0^{-1} w - \frac{1}{2} m_0^T \frac{\beta}{S_0} m_0 - \sum \frac{\beta}{2} t_N^2 + \sum \beta t_N w^T \phi(x_n) - \sum \frac{\beta}{2} w^T \phi(x_N) \phi(x_N)^T w\right) b_0^{a_0} \beta^{a_0 - 1 + \frac{N}{2}} e^{-b_0\beta}$

Quadratic $\rightarrow -\frac{1}{2} w^T \frac{\beta}{S_0} w + \sum \frac{\beta}{2} w^T \phi(x_N) \phi(x_N)^T w = -\frac{\beta}{2} w^T \left(S_0^{-1} + \sum^N \phi(x_N) \phi(x_N)^T\right) w$

Linear $\rightarrow \beta m_0^T S_0^{-1} w + \sum \beta t_N \phi(x_N)^T w = \beta\left(m_0^T S_0^{-1} + \sum t_N \phi(x_N)^T\right) w$

$m_N^T S_N^{-1} = m_0^T S_0^{-1} + \sum t_N \phi(x_N)^T$

$m_N^T = S_N \left(m_0^T S_0^{-1} + \sum t_N \phi(x_N)^T\right)$

Constant $\rightarrow -\sum^N \frac{\beta}{2} t_N^2 - \frac{\beta}{2} m_0^T S_0^{-1} m_0 = -\frac{\beta}{2}\left(\sum^N t_N^2 - m_0^T S_0^{-1} m_0\right)$

$p(w, \beta|t) = \frac{\beta}{S_N\sqrt{2\pi}} \exp\left(-\frac{\beta^2}{2S_N^2}(w - m_N)^2\right) \frac{b_N^{a_N}}{\Gamma(a)} \beta^{a_N-1} e^{-b_N\beta}$

$\propto \exp\left(-(w - m_N)^T \frac{\beta}{2S_N}(w - m_N)\right) b_N^{a_N} \beta^{a_N-1} e^{-b_N\beta}$

$\propto \exp\left(\underbrace{-\frac{\beta}{2} w^T S_N^{-1} w}_{\text{Quadratic}} \underbrace{+ \beta m_N^T S_N^{-1} w}_{\text{Linear}} \underbrace{- \frac{\beta}{2} m_N^T S_N^{-1} m_N}_{\text{Constant}}\right) b_N^{a_N} \beta^{a_N-1} e^{-b_N\beta}$

$b_0^{a_0} \beta^{a_0-1} \exp\left(-\frac{\beta}{2} m_0^T S_0^{-1} m_0 \underbrace{- \sum \frac{\beta}{2} t_N^2}_{\text{extra}} - b_0\beta\right)$

$\beta^{a_0 - 1 + \frac{N}{2}} = \beta^{a_N - 1}$

$a_N = a_0 + \frac{N}{2}$

$\exp\left(-\frac{\beta}{2} m_N^T S_N^{-1} m_N\right) b_N^{a_N} \beta^{a_N-1} \exp\left(-b_N\beta\right)$

$= b_N^{a_N} \beta^{a_N-1} \exp\left(-\frac{\beta}{2} m_N^T S_N^{-1} m_N - b_N\beta\right)$

$b_N = -b_0\beta - \frac{\beta}{2} m_0^T S_0^{-1} m_0 - \sum \frac{\beta}{2} t_N^2$

Q5

GMM formula

$$p(x, z \mid \mu, \Sigma) = \prod_{z} \pi_k N(x, z \mid \mu, \Sigma)^{z}$$

$$p(x \mid z) = \prod_{k=1}^{K} N(x \mid \mu_k, \Sigma)^{z_k}$$

$$p(z) = \prod_{k=1}^{k} \pi_k^{z_k}$$

$$L(x, z, \theta^{old}) = \prod_{i=1}^{N} p(z_i \mid \theta) p(x_i \mid \Sigma, \theta)$$

$$= \prod_{i=1}^{N} \prod_{k=1}^{k} N(x \mid \mu_k, \Sigma)^{z_{ik}} \pi_k^{z_{ik}} z_{ik}$$

$$\log(L) = \sum^{N} \sum^{k} \ln(N(x \mid \mu_k, \Sigma)^{z_{ik}} + \ln(\pi_k^{z_{ik}})$$

$$Q(\theta, \theta^{old}) = E[\log(L)] = \sum^{N} \sum^{k} E[z_{ik}]$$

$$= \sum^{N} \sum^{k} E[z_{ik}] \left(\ln(\pi_k^{z_{ik}}) - \frac{1}{2}(x - \mu_k)^T \Sigma^{-1} (x - \mu_k) - \frac{1}{2} \ln(\det(2\pi \Sigma)) \right)$$

$$\frac{\partial Q}{\partial \Sigma^{-1}} = \sum^{N} \sum^{k} E[z_{ik}] \left(-\frac{1}{2}(x - \mu_k)^T (x - \mu_k) - \frac{1}{2} \Sigma \right) = 0$$

$$\Sigma = \frac{1}{N} \sum^{N} \sum^{k} (x - \mu_k)^T (x - \mu_k) E[z_{ik}]$$

$$(x - \mu_k)^2$$

$$-2(x - \mu_k)$$

$$(2 - x)^2$$

$$-2(2 - x)$$

$$\frac{\partial Q}{\partial \mu} = \sum^{N} \sum^{K} E[z_{ik}] \mu_k = 0$$

$$\mu_k = \frac{1}{N} \sum^{N} \sum^{K} E[z_{ik}] \cdot x$$

Q6

ccc

Application Questions

In [ ]:
```python
# Imports
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import sklearn
from sklearn.mixture import GaussianMixture
from scipy.optimize import curve_fit
from scipy.stats import poisson as scipy_poisson, multivariate_normal, norm
```

Graphical Model 1

In [ ]:

```
# Part a)
probs = np.array([
        [1/2, 1/3, 1/4],
        [1/4, 1/3, 1/4],
        [1/4, 1/3, 1/2]
        ])

# "Cold" = 0, "Mild" = 1, "Hot" = 2
def draw_x_sample(n, print_vals=True):
    init = np.random.randint(3, size=1)
    sample_array = [init[0],]
    for i in range(0, n - 1):
        x_n_probs = probs[:, sample_array[-1]].T
        new = np.random.choice(a=[0,1,2], p=x_n_probs, size=1)
        sample_array.append(new[0])
    if print_vals:
        print(sample_array)
    return sample_array

for i in range(0,5):
    draw_x_sample(4)
```

```
[2, 2, 0, 0]
[0, 0, 0, 1]
[2, 1, 1, 1]
[2, 1, 1, 0]
[2, 2, 0, 2]
```

```
In [ ]: # Part b)
def print_probs(day, prob_list):
    print(day)
    print("\n".join("{} probability: {:.4f}".format(*i) for i in list(zip(["Co
ld", "Hot", "Mild"], prob_list))))



def marginal_probs(x0):
    day_probs = [x0]
    print_probs("X0", day_probs[0])
    for n in range(1,4):
        # p(x_1 = 0) = p(x_1=0|x_0=k)p(x_0=k)
        day_probs.append(
            [
                np.dot(day_probs[n-1], probs[0]),
                np.dot(day_probs[n-1], probs[1]),
                np.dot(day_probs[n-1], probs[2])
            ])
        print_probs(f"\nX{n}", day_probs[-1])
    return day_probs
marginal_prob = marginal_probs([1/3, 1/3, 1/3])
```

```
X0
Cold probability: 0.3333
Hot probability: 0.3333
Mild probability: 0.3333

X1
Cold probability: 0.3611
Hot probability: 0.2778
Mild probability: 0.3611

X2
Cold probability: 0.3634
Hot probability: 0.2731
Mild probability: 0.3634

X3
Cold probability: 0.3636
Hot probability: 0.2728
Mild probability: 0.3636
```

In [ ]:
```
# Part c)

x2 = [0,1,0]
print_probs("X2", x2)
print_probs("X3", probs.dot(x2))

# Find X1
x1_p = [0, 0, 0]
for i in range(3):
    p_val = (probs[i, :].ravel() / np.sum(probs, axis=1)[i])
    x1_p += p_val * x2[i]

print_probs("X1", x1_p)

x0_p = [0, 0, 0]
for i in range(3):
    p_val = (probs[i, :].ravel() / np.sum(probs, axis=1)[i])
    x0_p += p_val * x1_p[i]

print_probs("X0", x0_p)
```

```
X2
Cold probability: 0.0000
Hot probability: 1.0000
Mild probability: 0.0000
X3
Cold probability: 0.3333
Hot probability: 0.3333
Mild probability: 0.3333
X1
Cold probability: 0.3000
Hot probability: 0.4000
Mild probability: 0.3000
X0
Cold probability: 0.3277
Hot probability: 0.3446
Mild probability: 0.3277
```

In [ ]:
```
# Part d)
_="""
Going off of the results in part c, simply select the highest probabilities fo
r each day, so the most likely 4-day report would be:
Day 0: Hot
Day 1: Hot
Day 2: Hot
Day 3: Cold, Mild, or Hot
"""
```

Graphical Model 2

In [ ]:
```
# Part a)

def draw_samples_2(n, init_x=None):
    means = [-2, 0, 2]
    sample_x = init_x or [-2 if i == 0 else 0 if i == 1 else 2
                          for i in draw_x_sample(4, print_vals=False)]

    nd = lambda mu: np.random.normal(mu, 1, 1)[0]
    sample_y = [nd(i) for i in sample_x]

    # Convert to messages
    sample_x = ["Cold" if i == -2 else "Mild" if i == 0 else "Hot" for i in sa
mple_x]

    print("\n".join("{}: {}".format(*i) for i in list(zip(sample_x, sample_y
))))
    print()
    return sample_x, sample_y

for i in range(1, 6):
    print(f"Sample {i}:")
    draw_samples_2(4)
```

```
Sample 1:
Mild: 1.8472129094005176
Hot: 1.9428041558806037
Cold: -0.843895618832599
Hot: 3.0784412038506366

Sample 2:
Mild: -0.7364596635050101
Cold: -3.982106412733558
Mild: -0.08371721124696106
Cold: -4.348790566246894

Sample 3:
Mild: -0.5798886321513702
Mild: 0.054018596355834114
Cold: -0.6076958725766892
Cold: -2.741102815880099

Sample 4:
Hot: 2.5153761313634453
Hot: 0.5639943177430187
Cold: -3.02978457031303
Mild: -1.251069467755893

Sample 5:
Hot: 1.6651828018598722
Cold: -1.3908265915994769
Hot: 1.8847560217818158
Mild: 0.5670871907293726
```

In [ ]:

```
# Part b)
for i in range(5):
    draw_samples_2(n=4, init_x=[2, 0, -2, -2])
```

```
Hot: 2.699905309928331
Mild: -0.4334157820200868
Cold: -2.7368222024982964
Cold: -1.0584970882195466

Hot: 2.280338854129213
Mild: 0.04987847699197225
Cold: -1.9716674681208215
Cold: -2.608790348622337

Hot: 1.6572917950446817
Mild: -0.5343933389585854
Cold: -1.3690948388747337
Cold: -3.1113177909122216

Hot: 1.7208943097819538
Mild: -0.690784223522233
Cold: -0.5814091568866275
Cold: -3.1941833076035167

Hot: 1.2965292352830018
Mild: 0.1524303114414447
Cold: -2.490894873062886
Cold: -1.4301411072394252
```

In [ ]:

```
# Part c)
y_given = [0.7, 1.5, -1.8, -1]

np.zeros((4,3))
y_probs = np.array([
            norm.pdf(y_given, loc=-2, scale=1),
            norm.pdf(y_given, loc=-0, scale=1),
            norm.pdf(y_given, loc=2, scale=1)
            ]).T
y_probs[0] = y_probs[0] * 1/3
y_probs[1] = y_probs[0].dot(y_probs[1])
y_probs[2] = y_probs[1].dot(y_probs[2])
y_probs[3] = y_probs[2].dot(y_probs[3])

print(y_probs)
```

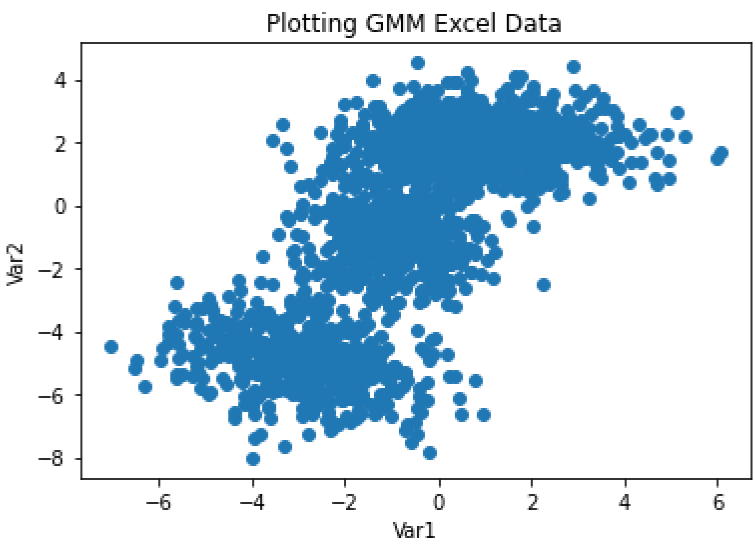
```
[[0.00347364 0.10408464 0.05712286]
 [0.0335948  0.0335948  0.0335948 ]
 [0.01579913 0.01579913 0.01579913]
 [0.00771587 0.00771587 0.00771587]]
```

Gaussian Mixture Model

In [ ]:

```
# 1. Visualization
df_gmm = pd.read_excel("gmm_data.xlsx")
print(df_gmm.head())

fig_gmm, ax_gmm = plt.subplots()
ax_gmm.scatter(df_gmm["Var1"], df_gmm["Var2"])
ax_gmm.set_xlabel("Var1")
ax_gmm.set_ylabel("Var2")
ax_gmm.set_title("Plotting GMM Excel Data")

_="""
Judging from the plot, one can argue there are three or four clusters in the d
ataset. The points in the upper part of the plot are all very close together b
ut the cavities on the top and bottom (around (1,0) and (1,3)) suggest those p
oints may be part of two clusters.
"""
```

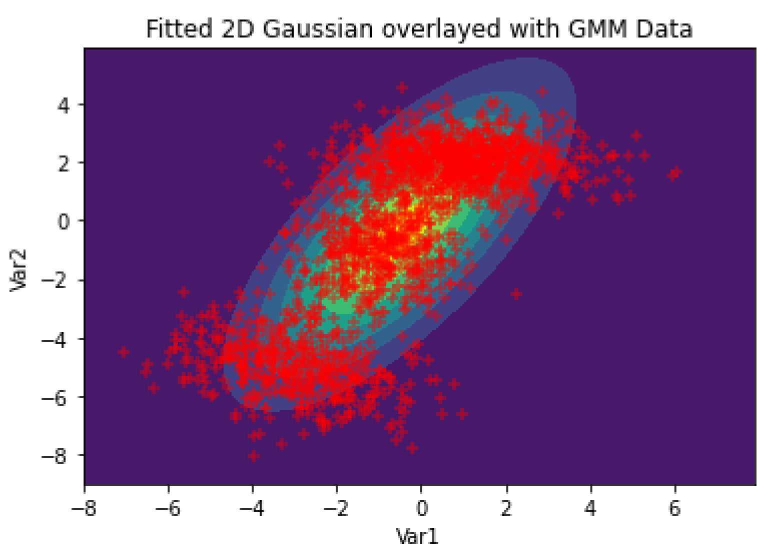
```
       Var1      Var2
0  2.915686  2.585758
1  1.923055  2.368691
2 -0.958997  1.834570
3 -3.563088 -2.487976
4 -0.626078 -0.120542
```



In [ ]:
```
# 2. Fitting 2D Normal Distribution
gaussian = lambda x, h, mu_x, mu_y, sigma: 1 / (sigma * np.sqrt(2 * np.pi)) *
np.exp(-0.5 * (x - mu)**2 / sigma**2)

gmm_x = df_gmm["Var1"]
gmm_y = df_gmm["Var2"]
mu = (gmm_x.mean(), gmm_y.mean())
covariance = np.cov(df_gmm[["Var1", "Var2"]], rowvar=False)

print(mu)
print(covariance)

x_mvg = np.linspace(min(gmm_x), max(gmm_x))
y_mvg = np.linspace(min(gmm_y), max(gmm_y))

mv = multivariate_normal(mean=mu, cov=covariance)
x, y = np.mgrid[-8:8:0.1, -9:6:0.1]
pdf_vals = mv.pdf(np.dstack((x, y)))

fig_mvn, ax_mvn = plt.subplots()
ax_mvn.contourf(x, y, pdf_vals)
ax_mvn.scatter(df_gmm["Var1"], df_gmm["Var2"], marker="+", color="red", alpha=
0.5)
ax_mvn.set_title("Fitted 2D Gaussian overlayed with GMM Data")
ax_mvn.set_xlabel("Var1")
ax_mvn.set_ylabel("Var2")
```

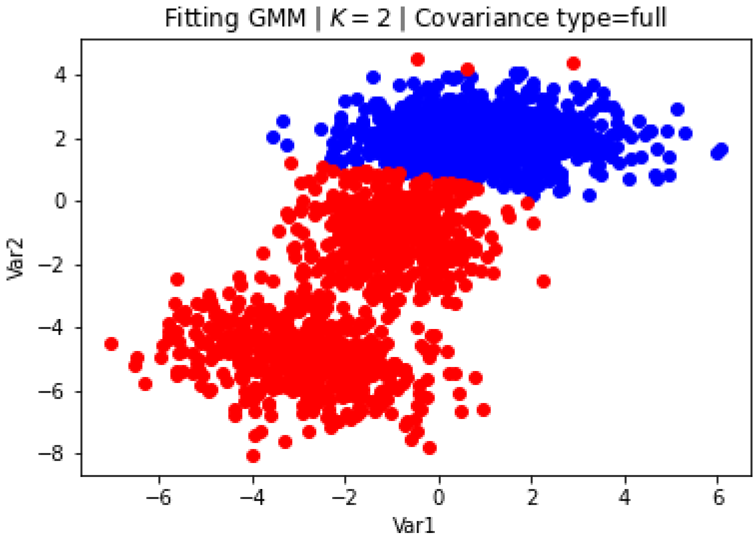
```
(-0.5002564557442571, -0.4773155803338159)
[[4.32492271 4.5356048 ]
 [4.5356048  9.01112111]]
```

Out[ ]:
```
Text(0, 0.5, 'Var2')
```

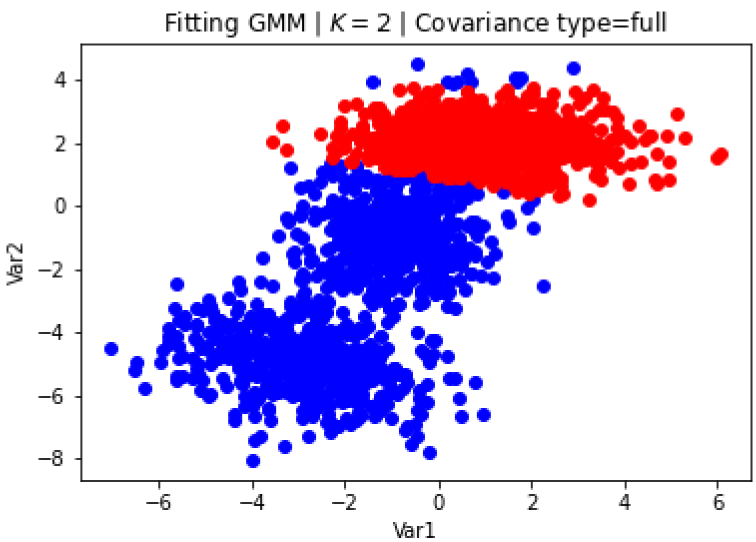


In [ ]:

```
# 3. Fitting GMM, K=2

def gmm_predict(k, mean_start=None, cov_type="full"):
    gmm = GaussianMixture(n_components=k, covariance_type=cov_type, means_init
=mean_start)
    labels = gmm.fit_predict(df_gmm)

    _, ax_gmm_fit = plt.subplots()

    def color(label_val):
        if label_val == 0:
            return "blue"
        if label_val == 1:
            return "red"
        if label_val == 2:
            return "green"
    for i in range(0,3):
        gm_x_n = [gmm_x[x_i] for x_i in range(len(gmm_x)) if labels[x_i] == i]
        gm_y_n = [gmm_y[y_i] for y_i in range(len(gmm_y)) if labels[y_i] == i]
        ax_gmm_fit.scatter(gm_x_n, gm_y_n, color=color(i))
    caption = f"Using initial means: {mean_start}"

    ax_gmm_fit.set_title(f"Fitting GMM | $K={k}$ | Covariance type={cov_type}"
)
    ax_gmm_fit.set_xlabel(f"Var1\n\n{caption}")
    ax_gmm_fit.set_ylabel("Var2")

gmm_predict(2, [(0,0), (1,-1)])
gmm_predict(2, [(1,1), (2,6)])

_="""
This run does not show much difference between the different starting means, b
ut run multiple times, they will show different clusters.
"""
```



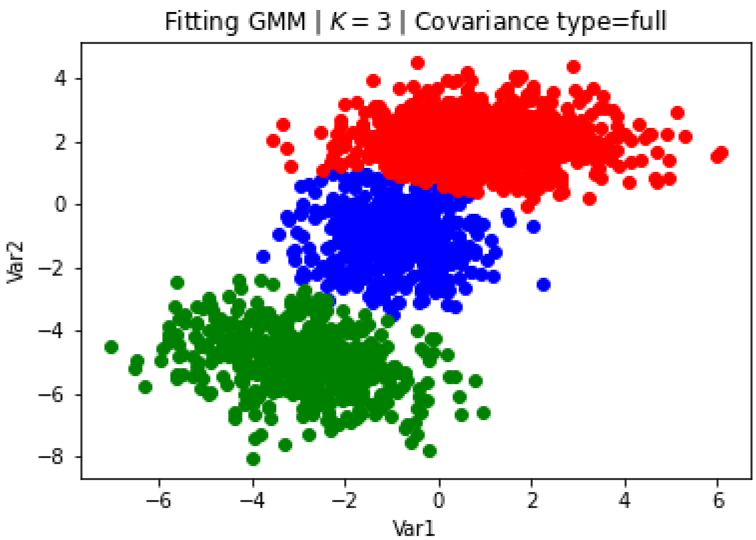
Using initial means: [(0, 0), (1, -1)]



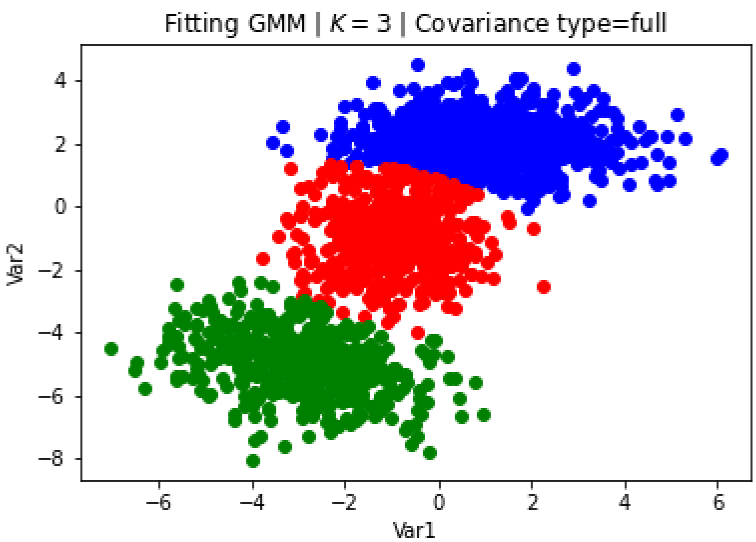
Using initial means: [(1, 1), (2, 6)]

In [ ]:

```
# 4. Fitting GMM, K=3

gmm_predict(3, [(0,0), (0,1), (-1,-1)])
gmm_predict(3, [(1,1), (-1,1), (-2,-1)])

_="""
Evidently, k=3 is a better model since the clusters appear more natural.
"""
```



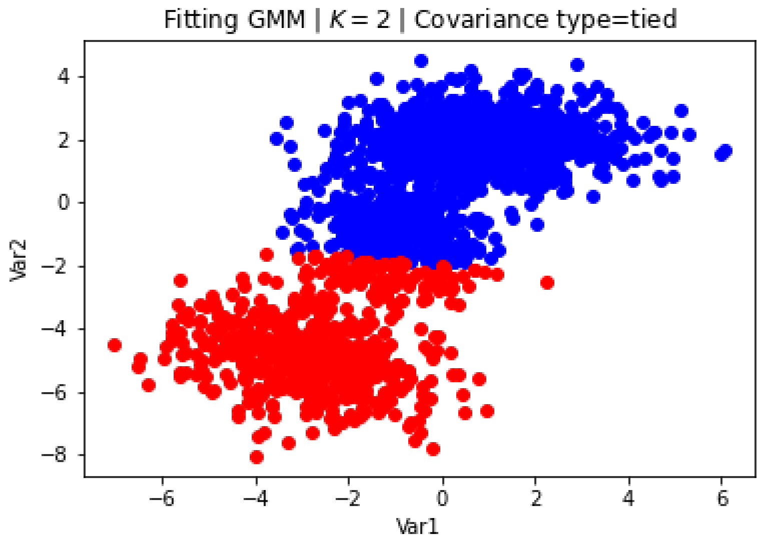
Using initial means: [(0, 0), (0, 1), (-1, -1)]



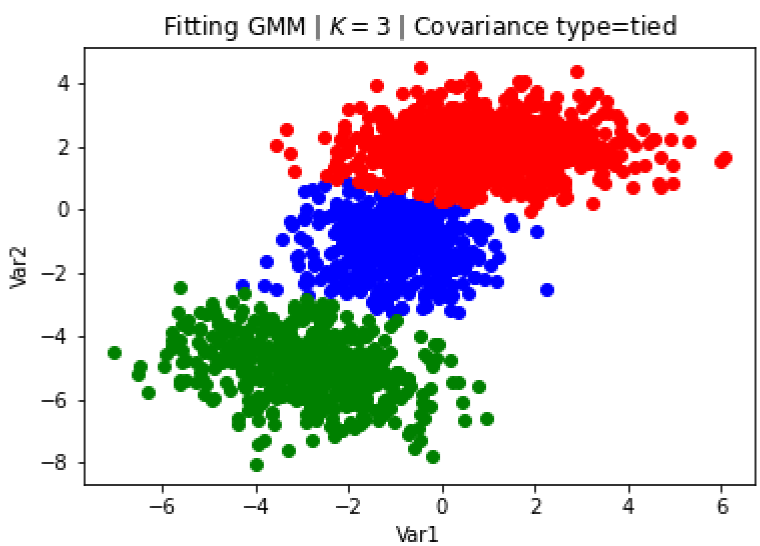
Using initial means: [(1, 1), (-1, 1), (-2, -1)]

In [ ]:

```python
# 5. Common covariance

gmm_predict(2, None, "tied")
gmm_predict(3, None, "tied")
```



Using initial means: None



Using initial means: None

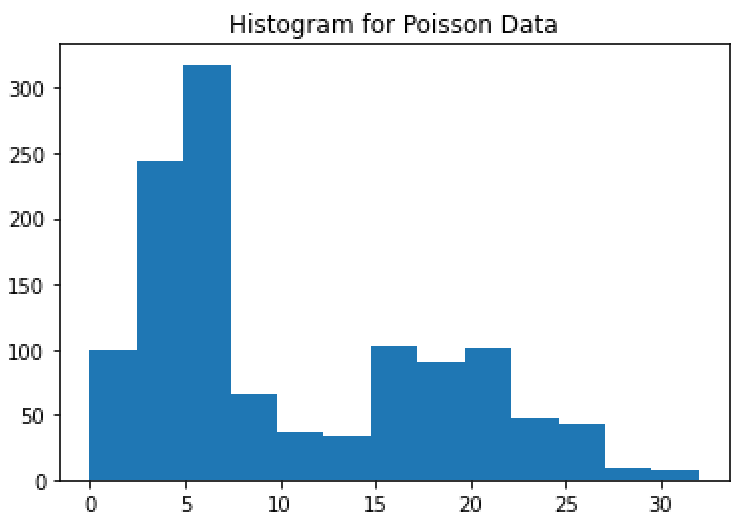
Poisson Mixture Model

In [ ]:

```python
# 1. Visualization

df_pmm = pd.read_excel("poisson_data.xlsx")
print(df_pmm.head())

fig_poisson, ax_poisson = plt.subplots()
ax_poisson.hist(df_pmm, bins="auto")
ax_poisson.set_title("Histogram for Poisson Data")
```

```
    X
0   2
1   6
2  18
3  20
4   8
```

Out[ ]: Text(0.5, 1.0, 'Histogram for Poisson Data')

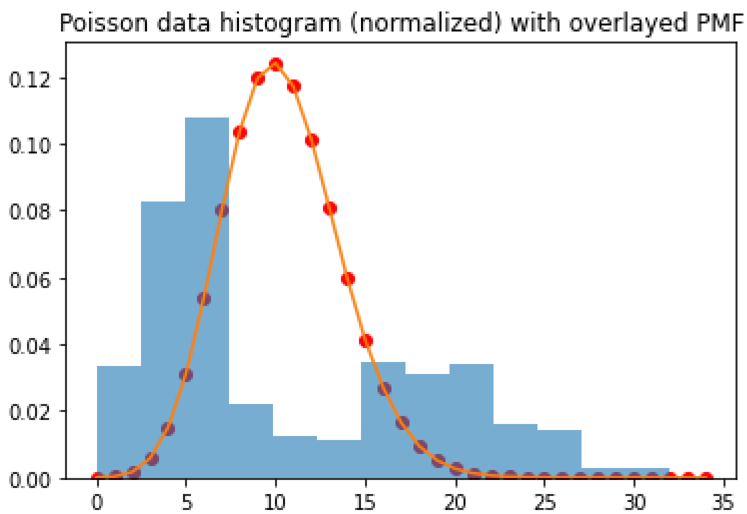


In [ ]:

```
# 2. Fit lambda with MLE

# The MLE estimate for lambda is simply the mean of the data
lambda_mle = df_pmm.mean()
poisson_x = np.arange(0, 35, 1)
_, ax_poisson_fit = plt.subplots()
poisson_mle = lambda x: (lambda_mle ** x * np.exp(-lambda_mle)) / np.math.fact
orial(x)
poisson_fitted_data = [poisson_mle(i) for i in poisson_x]

ax_poisson_fit.hist(df_pmm, density=True, alpha=0.6, bins="auto")
ax_poisson_fit.plot(poisson_x, poisson_fitted_data)
ax_poisson_fit.scatter(poisson_x, poisson_fitted_data, color="red")
ax_poisson_fit.set_title("Poisson data histogram (normalized) with overlayed P
MF")
```

Out[ ]:

```
Text(0.5, 1.0, 'Poisson data histogram (normalized) with overlayed PMF')
```



Part c - EM Update Rule

In [ ]:

```
def em_poisson(K, iters):
    weights = [0.5, 0.5]
    lambdas = np.random.random_integers(0, 30, size=K)

    # k is index, x_l is actual data
    q_k_l = lambda k, x_l: weights[k] * scipy_poisson(lambdas[k]).pmf(x_l)
    p_k_l = lambda k, x_l: q_k_l(k, x_l) / np.sum([q_k_l(a, x_l) for a in rang
e(K)])
    z_k = lambda k_i: np.sum(p_k_l(k_i, x) for x in df_pmm.to_numpy())
    N = df_pmm.size

    for _ in range(iters):
        # for p_k, lambda_k in list(zip(weights, lambdas)):
        for pi_k in range(K):
            lambdas[pi_k] = np.sum([x_l * p_k_l(pi_k, x_l) for x_l in df_pmm.t
o_numpy()]) / z_k(pi_k)
            weights[pi_k] = z_k(pi_k) / N
    return lambdas, weights

lambda_opt, weights_opt = em_poisson(K=2, iters=2)
print("Lambda vals", lambda_opt)
print("Mixture Weights", weights_opt)
fig_em, ax_em = plt.subplots()
ax_em.hist(df_pmm, density=True, alpha=0.6, bins="auto")
ax_em.scatter(poisson_x, weights_opt[0] * scipy_poisson(lambda_opt[0]).pmf(poi
sson_x), label=f"$\lambda={lambda_opt[0]}$")
ax_em.scatter(poisson_x, weights_opt[1] * scipy_poisson(lambda_opt[1]).pmf(poi
sson_x), label=f"$\lambda={lambda_opt[1]}$")
ax_em.set_title("Fitted Poisson Distribution with EM Algorithm")
ax_em.legend()
```

```
C:\Users\SAADMU~1\AppData\Local\Temp/ipykernel_10364/367784189.py:3: Deprecat
ionWarning: This function is deprecated. Please call randint(0, 30 + 1) inste
ad
  lambdas = np.random.random_integers(0, 30, size=K)
C:\Users\SAADMU~1\AppData\Local\Temp/ipykernel_10364/367784189.py:8: Deprecat
ionWarning: Calling np.sum(generator) is deprecated, and in the future will g
ive a different result. Use np.sum(np.fromiter(generator)) or the python sum
builtin instead.
  z_k = lambda k_i: np.sum(p_k_l(k_i, x) for x in df_pmm.to_numpy())

Lambda vals [ 5 19]
Mixture Weights [array([0.62589735]), array([0.37891899])]
```

Out[ ]:
```
<matplotlib.legend.Legend at 0x24909f69dc0>
```