**COEN 240 Machine Learning**

**Homework #4**

**Name:** Alex Cherekdjian **Student ID:** 00001083236

## Problem 1

1. $(3,0,2,1,3,2,1,0,2,1)$

$c_0 = 2$
$c_1 = 3$
$c_2 = 3$
$c_3 = 2$

$$MLE = \frac{\partial \ln P(data|\theta)}{\partial \theta} = 0$$

$$\ln(P(data|\theta)) = c_1 \ln \theta + c_0 \ln(1-\theta)$$

$$= \left(\tfrac{2}{3}\theta\right)^{c_0} \cdot \left(\tfrac{1}{3}\theta\right)^{c_1} \left(\tfrac{2}{3}(1-\theta)\right)^{c_2} \left(\tfrac{1}{3}(1-\theta)\right)^{c_3}$$

$$= \left(\tfrac{2\theta}{3}\right)^2 \left(\tfrac{1}{3}\theta\right)^3 \left(\tfrac{2}{3}(1-\theta)\right)^3 \left(\tfrac{1}{3}(1-\theta)\right)^2$$

$$= 2\left(\ln\left(\tfrac{2}{3}\right) + \ln\theta\right) + 3\left(\ln\left(\tfrac{1}{3}\right) + \ln(\theta)\right) + 3\left(\ln\left(\tfrac{2}{3}\right) + \ln(1-\theta)\right) + 2\left(\ln\left(\tfrac{1}{3}\right) + \ln(1-\theta)\right)$$

$$= 5\ln(\theta) + 2\ln\left(\tfrac{2}{3}\right) + 3\ln\left(\tfrac{1}{3}\right) + 5\ln(1-\theta) + 3\ln\left(\tfrac{2}{3}\right) + 2\ln\left(\tfrac{1}{3}\right)$$

$$= 5\ln(\theta) + 5\ln(1-\theta) + 2\ln\left(\tfrac{2}{3}\right) + 3\ln\left(\tfrac{1}{3}\right) + 3\ln\left(\tfrac{2}{3}\right) + 2\ln\left(\tfrac{1}{3}\right)$$

$$\frac{\partial \ln P(data|\theta)}{\partial \theta} = \frac{5}{\theta} - \frac{5}{1-\theta} = 0$$

$$\frac{1}{\theta} - \frac{1}{1-\theta} = 0$$

$$(1-\theta) - \theta = 0$$

$$1 - 2\theta = 0$$

$$\boxed{\theta = 1/2}$$

**Problem 2**

2. $$L(\theta) = \sum_{i=1}^{n} \ln f(x_i|\theta) = \sum_{i=1}^{n} \left( \ln\theta + \theta \ln x_0 - (\theta+1) \ln(x_i) \right)$$

$$= n \ln\theta + n\theta \ln x_0 - (\theta+1) \sum_{i=1}^{n} \ln(x_i)$$

$$\frac{\partial L(\theta)}{\partial \theta} = \frac{n}{\theta} + n \ln(x_0) - \sum_{i=1}^{n} \ln(x_i)$$

$$0 = \frac{n}{\theta} + n \ln(x_0) - \sum_{i=1}^{n} \ln(x_i)$$

$$0 = \frac{1}{\theta} + \ln(x_0) - \ln(x_i)$$

$$\ln(x) - \ln(x_0) = 1/\theta$$

$$\boxed{\theta = \frac{1}{\ln(x) - \ln(x_0)}}$$

**Problem 3**

3. $$P(\text{bus} | \text{late}) = \frac{P(\text{late} \,\&\, \text{bus})}{P(\text{late})} = \frac{P(\text{late} | \text{bus}) \cdot P(\text{bus})}{P(\text{late}|\text{bus}) \cdot P(\text{bus}) + P(\text{late}|\text{bike}) \cdot P(\text{bike})}$$

$P(\text{late} | \text{bus}) = 0.10$

$P(\text{late} | \text{bike}) = 0.02$

$P(\text{bike}) = 4/5$

$P(\text{bus}) = 1/5$

$$= \frac{0.1 \cdot (1/5)}{(0.1 \cdot 1/5) + (0.02 \cdot 4/5)}$$

$$\boxed{P(\text{bus}|\text{bike}) = 5/9}$$

**Problem 4**

4.1 $P_e = P(H_1)P(H_0|H_1) + P(H_0)P(H_1|H_0)$

decision rule $\frac{p(x_k|H_1)}{p(x_k|H_0)} \underset{H_0}{\overset{H_1}{\gtrless}} \gamma = \frac{P(H_0)}{P(H_1)}$

Proof: $P_e = P(H_1)P(H_0|H_1) + P(H_0)P(H_1|H_0)$

$P_e = P(H_1)\left[1 - P(H_1|H_1)\right] + P(H_0)P(H_1|H_0)$

$= P(H_1)\left[1 - \int_{R_1} p(x|H_1)dx\right] + P(H_0)\int_{R_1} p(x|H_0)\,dx$

$= P(H_1) + \int_{R_1}\left[P(H_0)p(x|H_0) - P(H_1)p(x|H_1)\right]dx$

$x$ in $R_1$ if:

$P(H_0)p(x|H_0) - P(H_1)p(x|H_1) < 0$

$P(H_1)p(x|H_1) \underset{H_0}{\overset{H_1}{\gtrless}} P(H_0)p(x|H_0)$

$$\boxed{\frac{p(x|H_1)}{p(x|H_0)} \underset{H_0}{\overset{H_1}{\gtrless}} \frac{P(H_0)}{P(H_1)} = \gamma}$$

4.2 $\max_{0 \le i \le M-1} P(H_i|x)$

$\max_{0 \le i \le M-1} \frac{P(H_i)P(x|H_i)}{P(x)} = \frac{P(H_i)P(x|H_i)}{\sum_{i=0}^{M-1} P(H_i)P(x|H_i)} \propto P(H_i)P(x|H_i)$

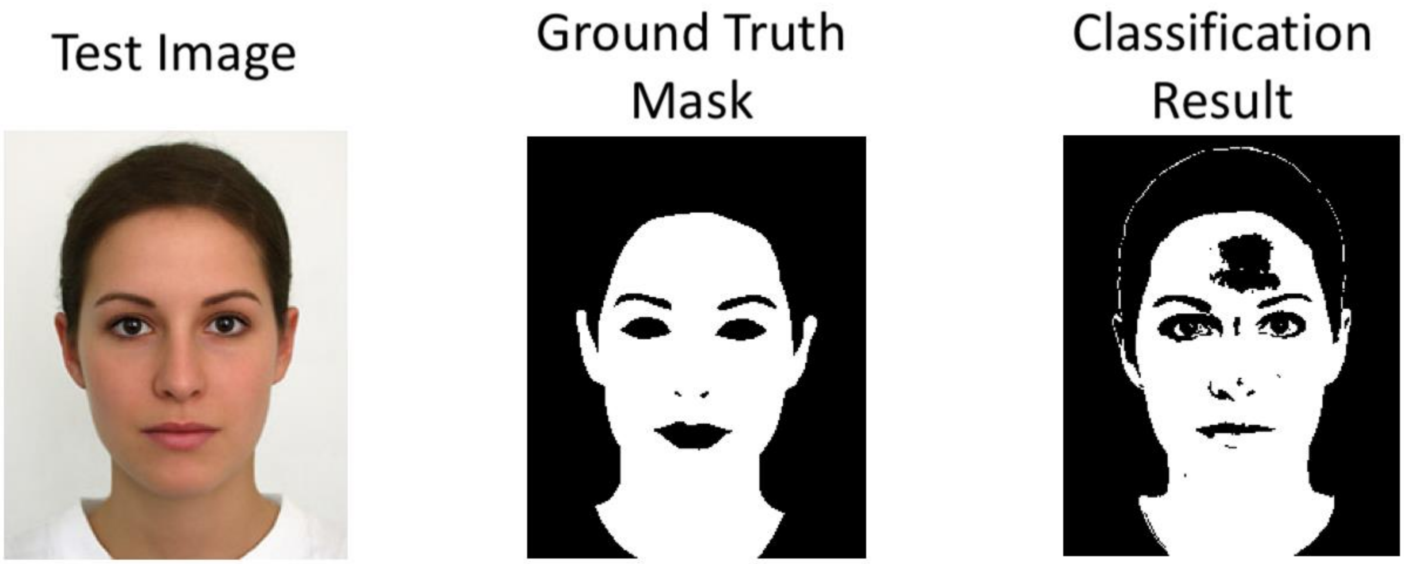
$$\boxed{\max_{i=0,1,\ldots M-1} P(H_i)P(x|H_i)}$$

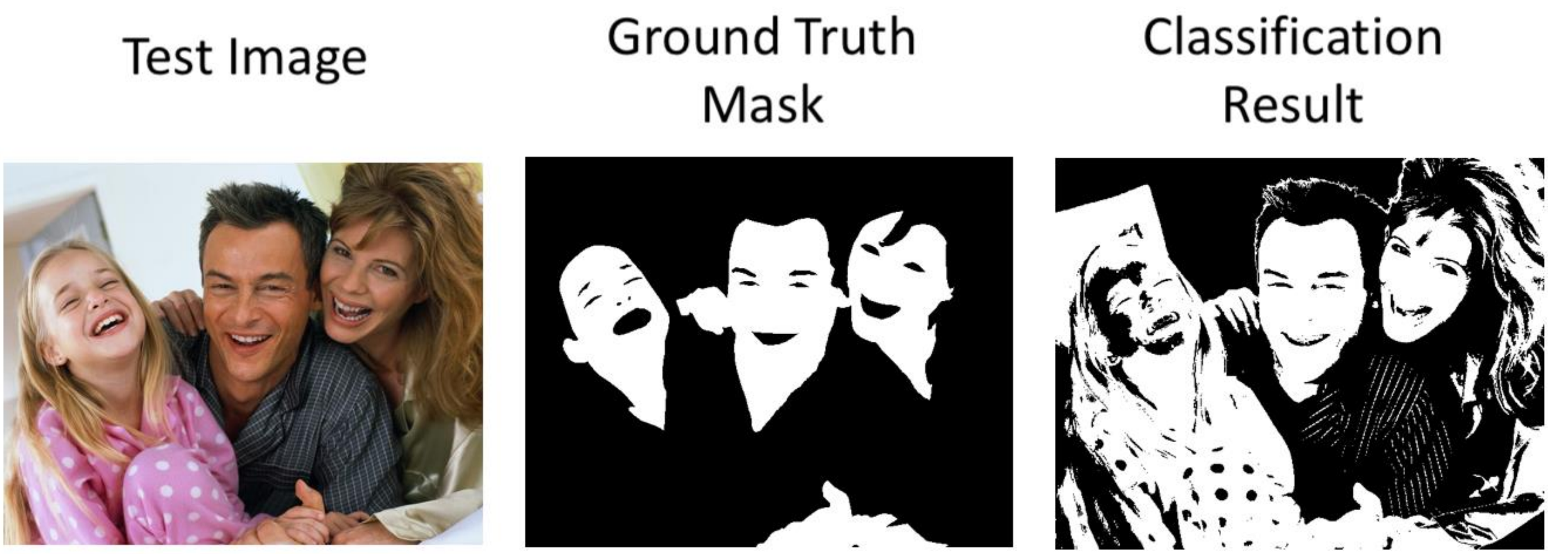
4.3 Naive Bayes Classifier assumes that all the features are independent of one another conditionally.

$$\max_{0 \le i \le M-1} P(H_i)\prod_{n=1}^{N} P(x_n|H_i)$$

Naive Bayes if with assumption

**Problem 5.1**




Comment: In the portrait picture, it is clearly shown that the algorithm does a fairly decent job differentiating between the background and skin despite a few mistakes. On the family picture, the separation between the skin pixels and background pixels get a little tough to separate due to the brighter colors in the picture that could pass as skin pixels. Despite the fact, the algorithm does a fairly decent job on both pictures.

**Problem 5.2**

Comment: The rates for the portrait and family picture are calculated in the code at the end. The results for each are tabulated below. The final calculations show how well the algorithm works on the portrait picture as well as how different the procedure works on the family picture.

Portrait:

true positive rate = 94.22

true negative rate = 94.62

false positive rate = 3.33

false negative rate = 9.33

Family:

true positive rate = 39.29

true negative rate = 89.31

false positive rate = 56.5

false negative rate = 11.49

**Attachment**

Problem 5 Code (in zip file): Code used for both family and portrait pictures. Just change the filename variable.

```
from PIL import Image
from math import sqrt, pi, exp

# to use code for portrait or family, change filename
filename = "family"

# open both real image and ground truth
im = Image.open(filename + ".jpg")
im_gnd = Image.open(filename + ".png")

# load pixels for real image and ground truth
px = im.load()
px_gnd = im_gnd.load()
```

```python
# pixels are skin constants ie.gnd=white
u_1r = 0
u_1g = 0
sigma_1r2 = 0
sigma_1g2 = 0
sigma_1r = 0
sigma_1g = 0
N_1 = 0

# pixels are background constants ie.gnd=black
u_0r = 0
u_0g = 0
sigma_0r2 = 0
sigma_0g2 = 0
sigma_0r = 0
sigma_0g = 0
N_0 = 0

# calculate u's and N's
for x in range(0, im.size[0]):
    for y in range(0, im.size[1]):

        # get pixel values of real image
        r_i = px[x,y][0]
        g_i = px[x,y][1]

        # get pixel values of ground truth
        r_i_gnd = px_gnd[x,y][0]
        g_i_gnd = px_gnd[x,y][1]

        if r_i_gnd == 0 and g_i_gnd == 0:
            # pixels are background
            u_0r += r_i
            u_0g += g_i
            N_0 += 1

        else:
            # pixels are skin
            u_1r += r_i
            u_1g += g_i
            N_1 += 1

u_0r /= N_0
u_0g /= N_0
u_1r /= N_1
```

```python
u_1g /= N_1

true_values_dict = {}

# calculate sigmas and sigmas^2
for x in range(0, im.size[0]):
    for y in range(0, im.size[1]):

        # get pixel values of real image
        r_i = px[x,y][0]
        g_i = px[x,y][1]

        # get pixel values of ground truth
        r_i_gnd = px_gnd[x,y][0]
        g_i_gnd = px_gnd[x,y][1]

        if r_i_gnd == 0 and g_i_gnd == 0:
            # pixels are background
            sigma_0r2 += (r_i - u_0r)**2
            sigma_0g2 += (g_i - u_0g)**2
            true_values_dict[x,y] = "background"

        else:
            # pixels are skin
            sigma_1r2 += (r_i - u_1r)**2
            sigma_1g2 += (g_i - u_1g)**2
            true_values_dict[x,y] = "skin"

sigma_0r2 /= N_0
sigma_0g2 /= N_0
sigma_1r2 /= N_1
sigma_1g2 /= N_1

sigma_0r = sqrt(sigma_0r2)
sigma_0g = sqrt(sigma_0g2)
sigma_1r = sqrt(sigma_1r2)
sigma_1g = sqrt(sigma_1g2)

# create new image for binary mask
im_mask = Image.new('RGB', (im.size[0], im.size[1]), color = 'black')
px_mask = im_mask.load()

predicted_values_dict = {}

# calculate probabilities
```

```
for x in range(0, im.size[0]):
    for y in range(0, im.size[1]):

        # get pixel values of real image
        r_k = px[x,y][0]
        g_k = px[x,y][1]

        # calculate first constants
        c0_r = 1/(sqrt(2*pi)*sigma_0r)
        c0_g = 1/(sqrt(2*pi)*sigma_0g)
        c1_r = 1/(sqrt(2*pi)*sigma_1r)
        c1_g = 1/(sqrt(2*pi)*sigma_1g)

        # calculate probabilities
        px_h0 = c0_r * exp(-0.5 * (((r_k - u_0r)**2)/sigma_0r2)) * c0_g *
exp(-0.5 * (((g_k - u_0g)**2)/sigma_0g2))
        px_h1 = c1_r * exp(-0.5 * (((r_k - u_1r)**2)/sigma_1r2)) * c1_g *
exp(-0.5 * (((g_k - u_1g)**2)/sigma_1g2))

        # if the pixel is skin, color pixel in mask white
        if (px_h1/px_h0) > (px_h0/px_h1):
            px_mask[x,y] = (255,255,255)
            predicted_values_dict[x,y] = "skin"
        else:
                predicted_values_dict[x,y] = "background"

# tp variables
true_skin = 0
predicted_skin = 0

# tn variables
true_background = 0
predicted_background = 0

# fp variables
predicted_skin_true_background = 0

# fp variables
predicted_background_true_skin = 0

# calculate rates
tp_rate = 0
tn_rate = 0
fp_rate = 0
fn_rate = 0
```

```python
# getting other values
for x in range(0, im.size[0]):
        for y in range(0, im.size[1]):

                # get pixel values of real image
                if predicted_values_dict[x,y] == "background":
                        # background else skin
                        true_background += 1
                else:
                        true_skin += 1

                if (true_values_dict[x,y] == "skin") and
(predicted_values_dict[x,y] == "skin"):
                        # for tp
                        predicted_skin += 1

                elif(true_values_dict[x,y] == "background") and
(predicted_values_dict[x,y] == "background"):
                        # for tn
                        predicted_background += 1

                elif(true_values_dict[x,y] == "background") and
(predicted_values_dict[x,y] == "skin"):
                        # for fp
                        predicted_skin_true_background += 1

                elif(true_values_dict[x,y] == "skin") and
(predicted_values_dict[x,y] == "background"):
                        # for fn
                        predicted_background_true_skin +=1

tp_rate = predicted_skin / true_skin * 100
tn_rate = predicted_background / true_background * 100
fp_rate = predicted_skin_true_background / true_background * 100
fn_rate = predicted_background_true_skin /true_skin * 100

print("true positive rate = " + str(round(tp_rate, 2)))
print("true negative rate = " + str(round(tn_rate, 2)))
print("false positive rate = " + str(round(fp_rate, 2)))
print("false negative rate = " + str(round(fn_rate, 2)))

# save the final binary mask
im_mask.save(filename + '_mask.png')
```