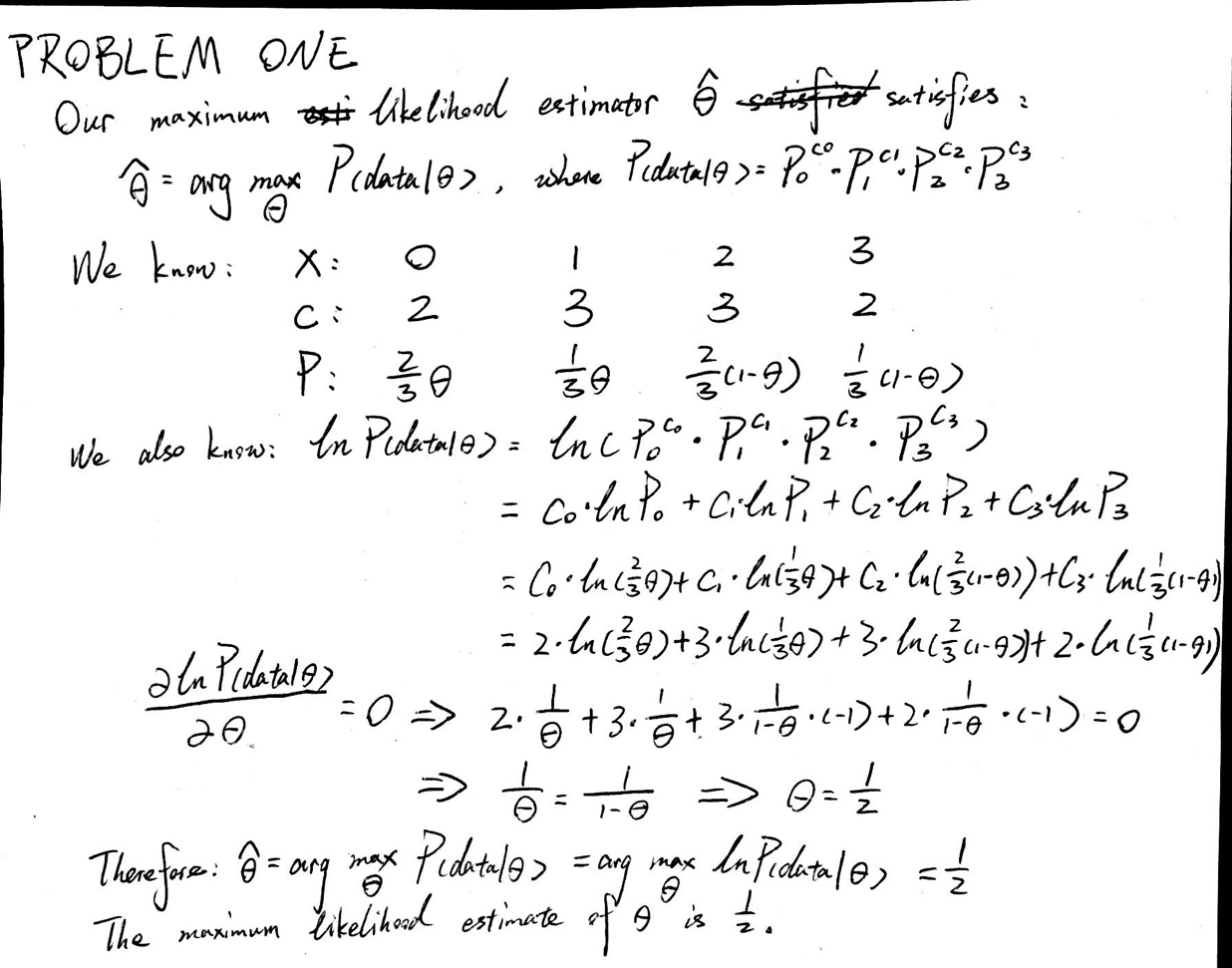
**COEN 240 MACHINE LEARNING**

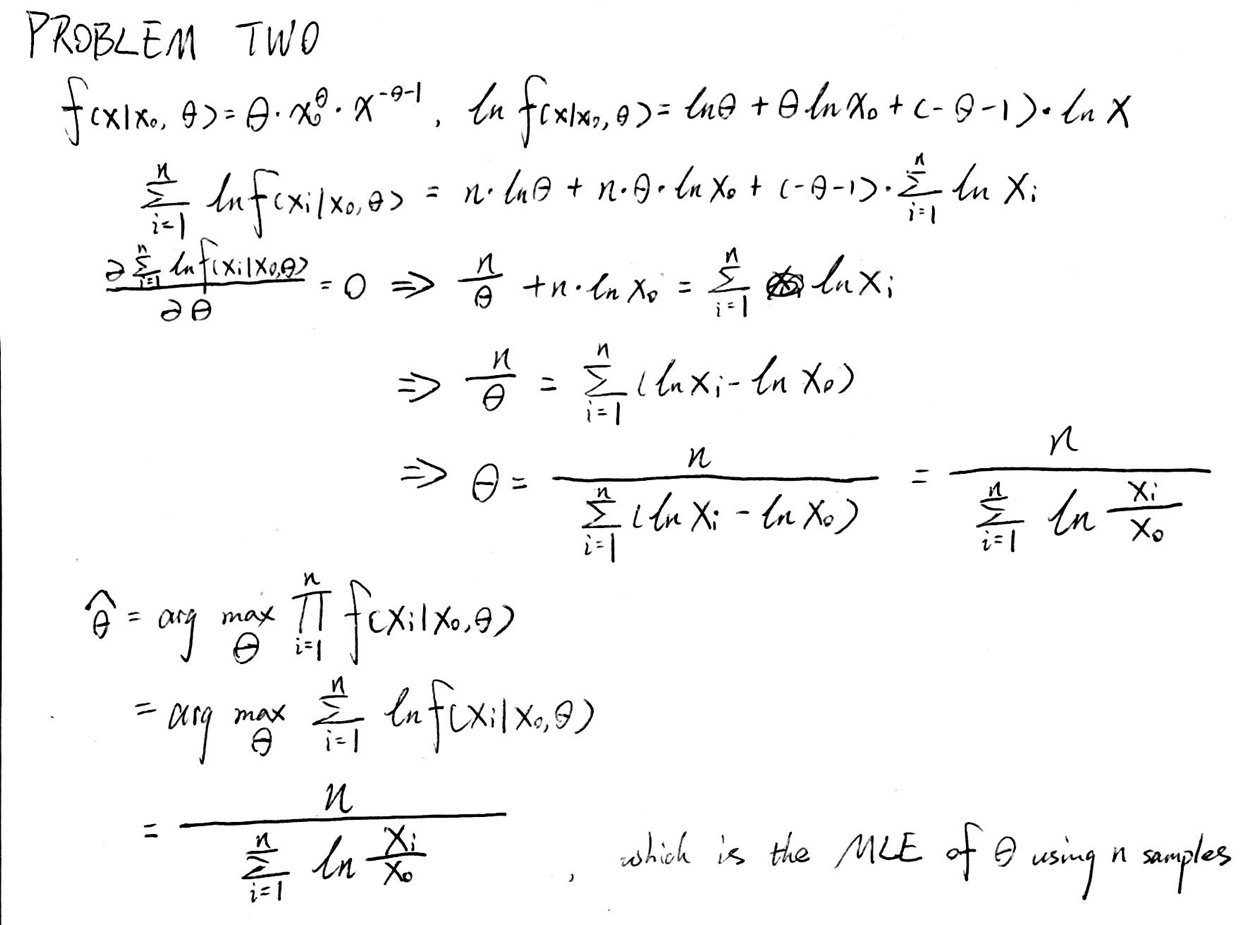
**HOMEWROK THREE**

**NAME: BOSEN YANG STUDENT ID: 1589880**

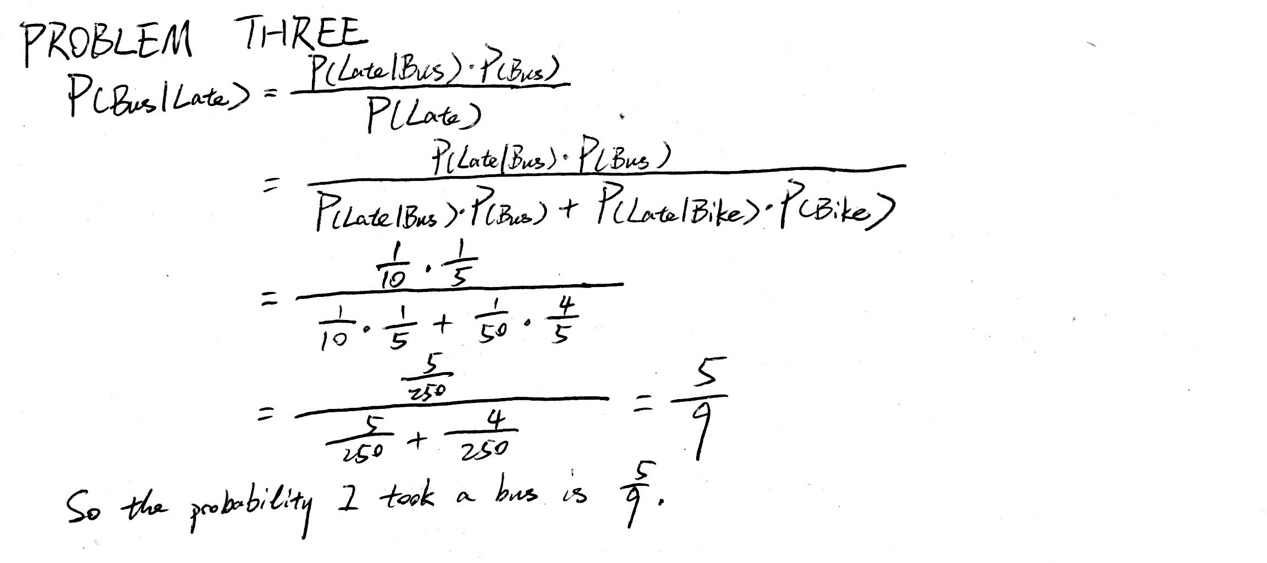
**PROBLEM ONE**



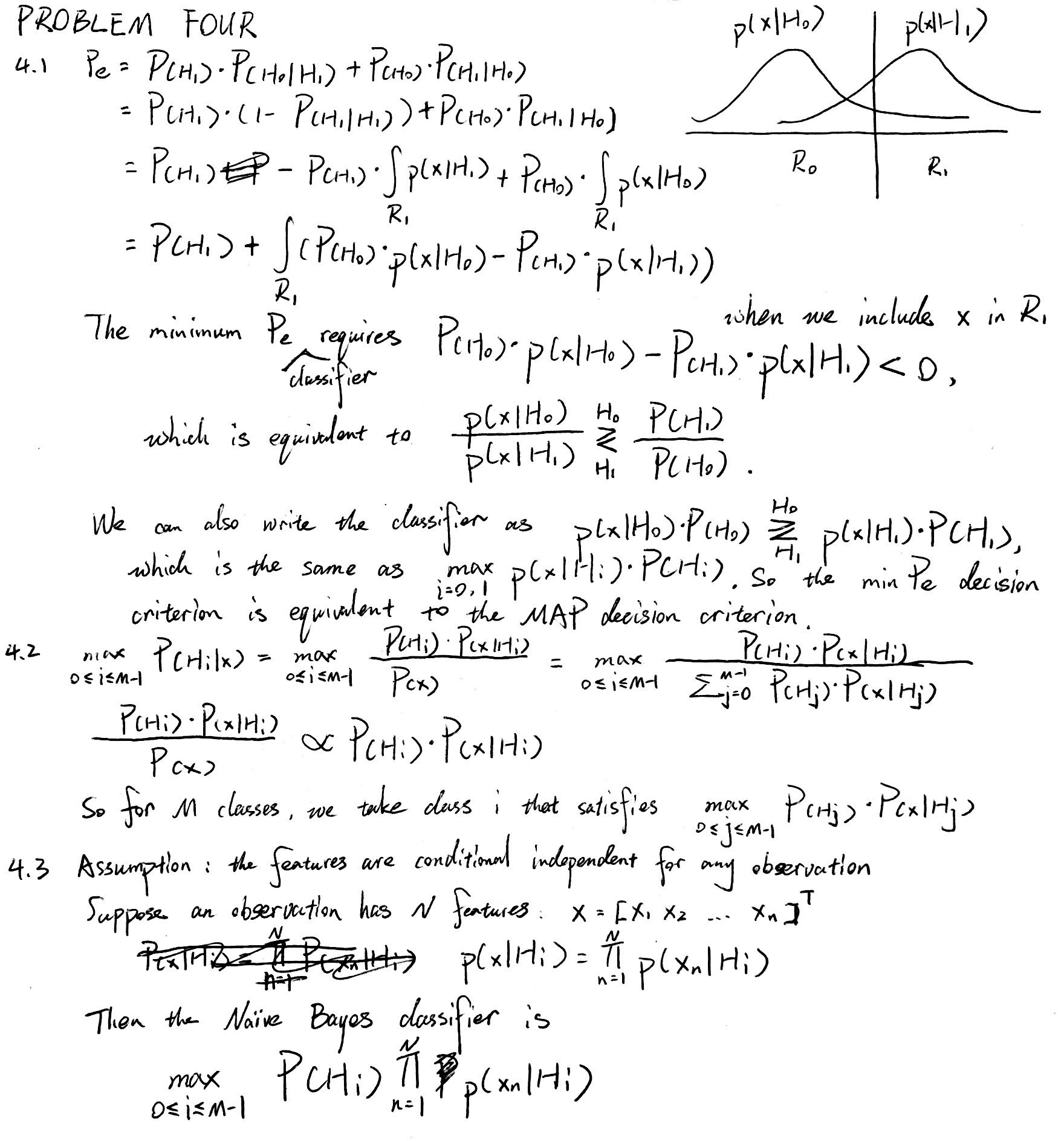
**PROBLEM TWO:**



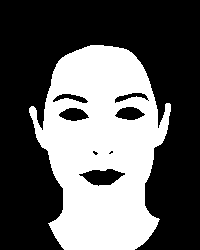
**PROBLEM THREE:**

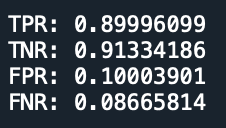


**PROBLEM FOUR**



**PROBLEM FIVE**

**  **

****

**ATTACHMENTS**

**PROBLEM ONE CODE**

"""

Created on Sun Jan 26 17:12:27 2020

@author: burson

"""

﻿#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Mon Feb 24 08:42:29 2020

@author: Burson

"""

import math

import numpy as np

from PIL import Image

from matplotlib.pyplot import imread

# INPUT READING ------------------------------------------

train\_image = imread("family.jpg")

train\_truth = imread("family.png")

test\_image = imread("portrait.jpg")

test\_truth = imread("portrait.png")

# INPUT PROCESSING ------------------------------------------

# color code extraction AND scale invariant transformation

train\_shape = train\_image.shape

train\_image\_flat = train\_image.reshape((train\_shape[0]\*train\_shape[1], train\_shape[2]))

train\_image\_base = np.sum(train\_image\_flat, axis=1) + 0.000000000000000001 # deal with 0's

train\_image\_flat\_trans = np.transpose(train\_image\_flat)

train\_r = np.divide(train\_image\_flat\_trans[0], train\_image\_base)

train\_g = np.divide(train\_image\_flat\_trans[1], train\_image\_base)

# deal with 0 elements

pos\_zero = np.argwhere(np.sum(train\_image\_flat, axis=1) == 0)

for i in pos\_zero:

train\_r[int(i[0])] = 1/3

train\_g[int(i[0])] = 1/3

del(train\_image, train\_image\_base, train\_image\_flat\_trans, pos\_zero)

# skin background split

train\_label\_s = train\_truth.reshape((train\_shape[0]\*train\_shape[1], train\_shape[2]+1)).transpose()[1]

train\_label\_b = 1-train\_label\_s

train\_r\_s = train\_r[np.argwhere(np.multiply(train\_r, train\_label\_s))]

train\_r\_b = train\_r[np.argwhere(np.multiply(train\_r, train\_label\_b))]

train\_g\_s = train\_g[np.argwhere(np.multiply(train\_g, train\_label\_s))]

train\_g\_b = train\_g[np.argwhere(np.multiply(train\_g, train\_label\_b))]

# prior probability calculation

pp\_s = np.count\_nonzero(train\_label\_s)/train\_image\_flat.shape[0]

pp\_b = np.count\_nonzero(train\_label\_b)/train\_image\_flat.shape[0]

del(train\_label\_s, train\_label\_b)

# color code extraction AND scale invariant transformation

test\_shape = test\_image.shape

test\_image\_flat = test\_image.reshape((test\_shape[0]\*test\_shape[1], test\_shape[2]))

test\_image\_base = np.sum(test\_image\_flat, axis=1) + 0.000000000000000001

test\_image\_flat\_trans = np.transpose(test\_image\_flat)

test\_r = np.divide(test\_image\_flat\_trans[0], test\_image\_base)

test\_g = np.divide(test\_image\_flat\_trans[1], test\_image\_base)

# deal with 0 elements

pos\_zero = np.argwhere(np.sum(test\_image\_flat, axis=1) == 0)

for i in pos\_zero:

test\_r[int(i[0])] = 1/3

test\_g[int(i[0])] = 1/3

test\_label\_s = test\_truth.reshape((test\_shape[0]\*test\_shape[1], test\_shape[2]+1)).transpose()[1]

del(test\_image, test\_image\_base, test\_image\_flat\_trans)

del(train\_truth, test\_truth)

# MODEL TRAINING ------------------------------------------

# use the closed-form solution we derived

miu\_r\_s = train\_r\_s.mean()

miu\_g\_s = train\_g\_s.mean()

var\_r\_s = train\_r\_s.var()

var\_g\_s = train\_g\_s.var()

miu\_r\_b = train\_r\_b.mean()

miu\_g\_b = train\_g\_b.mean()

var\_r\_b = train\_r\_b.var()

var\_g\_b = train\_g\_b.var()

# OUTPUT GENERATION ------------------------------------------

# calculate p(x|Hs) for skin

power\_s\_r = -np.square(test\_r - miu\_r\_s)/(2\*var\_r\_s)

power\_s\_g = -np.square(test\_g - miu\_g\_s)/(2\*var\_g\_s)

p\_Hs\_r = np.exp(power\_s\_r) / (math.sqrt(2\*np.pi\*var\_r\_s))

p\_Hs\_g = np.exp(power\_s\_g) / (math.sqrt(2\*np.pi\*var\_g\_s))

p\_Hs = np.multiply(p\_Hs\_r, p\_Hs\_g)

# calculate p(x|Hb) for background

power\_b\_r = -np.square(test\_r - miu\_r\_b)/(2\*var\_r\_b)

power\_b\_g = -np.square(test\_g - miu\_g\_b)/(2\*var\_g\_b)

p\_Hb\_r = np.exp(power\_b\_r) / (math.sqrt(2\*np.pi\*var\_r\_b))

p\_Hb\_g = np.exp(power\_b\_g) / (math.sqrt(2\*np.pi\*var\_g\_b))

p\_Hb = np.multiply(p\_Hb\_r, p\_Hb\_g)

del(test\_r, test\_g)

del(power\_s\_r, power\_s\_g, power\_b\_r, power\_b\_g, p\_Hs\_r, p\_Hs\_g, p\_Hb\_r, p\_Hb\_g)

# result generation applying MAP criterion

result\_s = (pp\_s\*p\_Hs - pp\_b\*p\_Hb) > 0

# ditected binary mask generation

ones = 255\*np.multiply(np.ones(test\_image\_flat.shape[0]), result\_s)

result\_array = []

for i in range(3):

result\_array.append(ones)

result\_array = np.array(result\_array).transpose().reshape((test\_shape[0], test\_shape[1], test\_shape[2]))

result\_array = result\_array.astype(np.uint8)

result\_image = Image.fromarray(result\_array)

result\_image.save("result.png")

del(i, ones, result\_image, result\_array)

# OUTPUT EVALUATION ------------------------------------------

num\_positive = np.count\_nonzero(result\_s)

num\_negative = test\_image\_flat.shape[0] - num\_positive

true\_match\_s = np.count\_nonzero(np.multiply(test\_label\_s, result\_s))

true\_match\_b = np.count\_nonzero(np.multiply(1-test\_label\_s, 1-result\_s))

false\_match\_s = np.count\_nonzero(np.multiply(1-test\_label\_s, result\_s))

false\_match\_b = np.count\_nonzero(np.multiply(test\_label\_s, 1-result\_s))

tpr = true\_match\_s / num\_positive

tnr = true\_match\_b / num\_negative

fpr = false\_match\_s / num\_positive

fnr = false\_match\_b / num\_negative

print("\n\n")

print("TPR: %.8f\nTNR: %.8f" % (tpr, tnr))

print("FPR: %.8f\nFNR: %.8f" % (fpr, fnr))

print("\n\n")