COEN 240 MACHINE LEARNING

HOMEWROK THREE

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

PROBLEM ONE

PROBLEM ONE

Our maximum ~~esti~~ likelihood estimator $\hat{\theta}$ ~~satisfied~~ satisfies:

$$\hat{\theta} = \arg\max_{\theta} P(data|\theta), \quad \text{where } P(data|\theta) = P_0^{c_0} \cdot P_1^{c_1} \cdot P_2^{c_2} \cdot P_3^{c_3}$$

We know:

X:	0	1	2	3
c:	2	3	3	2
P:	$\frac{2}{3}\theta$	$\frac{1}{3}\theta$	$\frac{2}{3}(1-\theta)$	$\frac{1}{3}(1-\theta)$

We also know:

$$\begin{aligned}
\ln P(data|\theta) &= \ln(P_0^{c_0} \cdot P_1^{c_1} \cdot P_2^{c_2} \cdot P_3^{c_3}) \\
&= c_0 \cdot \ln P_0 + c_1 \cdot \ln P_1 + c_2 \cdot \ln P_2 + c_3 \cdot \ln P_3 \\
&= c_0 \cdot \ln(\tfrac{2}{3}\theta) + c_1 \cdot \ln(\tfrac{1}{3}\theta) + c_2 \cdot \ln(\tfrac{2}{3}(1-\theta)) + c_3 \cdot \ln(\tfrac{1}{3}(1-\theta)) \\
&= 2 \cdot \ln(\tfrac{2}{3}\theta) + 3 \cdot \ln(\tfrac{1}{3}\theta) + 3 \cdot \ln(\tfrac{2}{3}(1-\theta)) + 2 \cdot \ln(\tfrac{1}{3}(1-\theta))
\end{aligned}$$

$$\frac{\partial \ln P(data|\theta)}{\partial \theta} = 0 \Rightarrow 2 \cdot \frac{1}{\theta} + 3 \cdot \frac{1}{\theta} + 3 \cdot \frac{1}{1-\theta} \cdot (-1) + 2 \cdot \frac{1}{1-\theta} \cdot (-1) = 0$$

$$\Rightarrow \frac{1}{\theta} = \frac{1}{1-\theta} \Rightarrow \theta = \frac{1}{2}$$

Therefore: $\hat{\theta} = \arg\max_{\theta} P(data|\theta) = \arg\max_{\theta} \ln P(data|\theta) = \frac{1}{2}$

The maximum likelihood estimate of θ is $\frac{1}{2}$.

PROBLEM TWO:

PROBLEM TWO

$f(x|x_0,\theta) = \theta \cdot x_0^{\theta} \cdot x^{-\theta-1}$, $\ln f(x|x_0,\theta) = \ln\theta + \theta \ln x_0 + (-\theta-1)\cdot \ln x$

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

$$\frac{\partial \sum_{i=1}^{n}\ln f(x_i|x_0,\theta)}{\partial\theta} = 0 \Rightarrow \frac{n}{\theta} + n\cdot\ln x_0 = \sum_{i=1}^{n}\ln x_i$$

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

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

$$\hat{\theta} = \arg\max_{\theta} \prod_{i=1}^{n} f(x_i|x_0,\theta)$$

$$= \arg\max_{\theta} \sum_{i=1}^{n}\ln f(x_i|x_0,\theta)$$

$$= \frac{n}{\sum_{i=1}^{n}\ln\frac{x_i}{x_0}}$$, which is the MLE of θ using n samples

PROBLEM THREE:

PROBLEM THREE

$$P(Bus|Late) = \frac{P(Late|Bus)\cdot P(Bus)}{P(Late)}$$

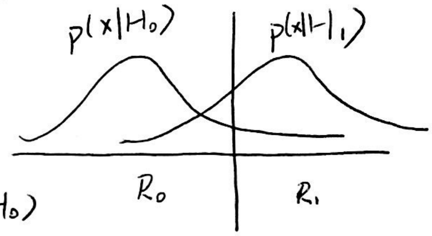
$$= \frac{P(Late|Bus)\cdot P(Bus)}{P(Late|Bus)\cdot P(Bus) + P(Late|Bike)\cdot P(Bike)}$$

$$= \frac{\frac{1}{10}\cdot\frac{1}{5}}{\frac{1}{10}\cdot\frac{1}{5} + \frac{1}{50}\cdot\frac{4}{5}}$$

$$= \frac{\frac{5}{250}}{\frac{5}{250} + \frac{4}{250}} = \frac{5}{9}$$

So the probability I took a bus is $\frac{5}{9}$.

PROBLEM FOUR



PROBLEM FOUR

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

$= P(H_1)\cdot(1 - P(H_1|H_1)) + P(H_0)\cdot P(H_1|H_0)$

$= P(H_1)\cancel{P} - P(H_1)\cdot\int_{R_1} p(x|H_1) + P(H_0)\cdot\int_{R_1} p(x|H_0)$

$= P(H_1) + \int_{R_1} (P(H_0)\cdot p(x|H_0) - P(H_1)\cdot p(x|H_1))$

The minimum P_e classifier requires $P(H_0)\cdot p(x|H_0) - P(H_1)\cdot p(x|H_1) < 0$, when we include x in R_1

which is equivalent to $\dfrac{p(x|H_0)}{p(x|H_1)} \underset{H_1}{\overset{H_0}{\gtrless}} \dfrac{P(H_1)}{P(H_0)}$.

We can also write the classifier as $p(x|H_0)\cdot P(H_0) \underset{H_1}{\overset{H_0}{\gtrless}} p(x|H_1)\cdot P(H_1)$, which is the same as $\max_{i=0,1} p(x|H_i)\cdot P(H_i)$. So the min P_e decision criterion is equivalent to the MAP decision criterion.

4.2 $\max_{0\le i\le M-1} P(H_i|x) = \max_{0\le i\le M-1} \dfrac{P(H_i)\cdot P(x|H_i)}{P(x)} = \max_{0\le i\le M-1} \dfrac{P(H_i)\cdot P(x|H_i)}{\sum_{j=0}^{M-1} P(H_j)\cdot P(x|H_j)}$

$\dfrac{P(H_i)\cdot P(x|H_i)}{P(x)} \propto P(H_i)\cdot P(x|H_i)$

So for M classes, we take class i that satisfies $\max_{0\le j\le M-1} P(H_j)\cdot P(x|H_j)$

4.3 Assumption: the features are conditional independent for any observation

Suppose an observation has N features: $x = [x_1\ x_2\ \dots\ x_n]^T$

$\cancel{P(x|H_i) = \prod_{n=1}^{N} P(x_n|H_i)}$ $p(x|H_i) = \prod_{n=1}^{N} p(x_n|H_i)$

Then the Naïve Bayes classifier is

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

PROBLEM FIVE

```
TPR: 0.89996099
TNR: 0.91334186
FPR: 0.10003901
FNR: 0.08665814
```

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")
```