Problem 1

Q1.

The prior of C_1 is 0.34125, C_2 is 0.33833 and C_3 is 0.32042.

```
total_num = len(input1_train)
pC1 = len(classes[1])/total_num
pC2 = len(classes[2])/total_num
pC3 = len(classes[3])/total_num
pCi = [pC1, pC2, pC3]
print("pCi value: ",pCi)
```

```
pCi value:  [0.34125, 0.3383333333333333, 0.3204166666666667]
```

Q2.

p(x=1|C1) $= {p_1}^x \cdot (1 - p_1)^{1-x}$
$= p_1$
= 0.34432

p(x=1|C2) $= {p_2}^x \cdot (1 - p_2)^{1-x}$
$= p_2$
= 0.61207

p(x=1|C2) $= {p_3}^x \cdot (1 - p_3)^{1-x}$
$= p_3$
= 0.40182

```
p1 = (classes[1]['feature_value']==1).sum()/len(classes[1])
p2 = (classes[2]['feature_value']==1).sum()/len(classes[2])
p3 = (classes[3]['feature_value']==1).sum()/len(classes[3])
pi = [p1, p2, p3]
print("pi value: ",pi)
```

```
pi value:  [0.3443223443223443, 0.6120689655172413, 0.40182054616384916]
```

CSCI3320 Programming Homework 0 Report **Chung Tsz Ting / 1155110208**

Q3.

My discriminant function is $\log p_i{}^x + \log(1 - p_i)^{1-x} + \log P(C_i)$ for prior $P(C_i)$ and the log likelihood $\log p_i{}^x + \log(1 - p_i)^{1-x}$.

```
def confusion_matrix():
    #pirj = predicted result i and actual result j
    p1r1, p1r2, p1r3, p2r1, p2r2, p2r3, p3r1, p3r2, p3r3 = 0,0,0,0,0,0,0,0,0
    for i in range(len(y_test)):
        if(y_test.iloc[i]==1):
            if(result[i]==1): p1r1+=1
            elif(result[i]==2): p2r1+=1
            else: p3r1+=1
        elif(y_test.iloc[i]==2):
            if(result[i]==1): p1r2+=1
            elif(result[i]==2): p2r2+=1
            else: p3r2+=1
        else:
            if(result[i]==1): p1r3+=1
            elif(result[i]==2): p2r3+=1
            else: p3r3+=1
    print('\t\t\033[96mpredict\033[0;0m\t')
    print('\t\t\033[96m1\t2\t3\033[0;0m')
    print('\t\033[96m1\033[0;0m\t'+'\033[1m'+str(p1r1)+'\t'+str(p2r1)+'\t'+str(p3r1)+'\033[0;0m')
    print('\033[96mactual\t2\033[0;0m\t'+'\033[1m'+str(p1r2)+'\t'+str(p2r2)+'\t'+str(p3r2)+'\033[0;0m')
    print('\t\033[96m3\033[0;0m\t'+'\033[1m'+str(p1r3)+'\t'+str(p2r3)+'\t'+str(p3r3)+'\033[0;0m')
    return "\033[96m---------confusion_matrix---------"
print(confusion_matrix())
print("acc: ",(y_test==result).sum()/len(y_test))
```

```
                predict
                1       2       3
        1       135     70      0
actual  2       70      108     0
        3       148     69      0
---------confusion_matrix---------
acc:  0.405
```

Confusion matrix is as above and along the row is the number of instances in predicted class i while along the column is the the number of instances in actual class i for 1 ≤ i ≤ 3.

Q4. accuracy = 40.5%

As precision = TP/TP+FP,

precision for class 1 = 135 / (135+70+148)
= 0.38244

precision for class 2 = 108 / (70+108+69)
= 0.43725

precision for class 3 = 0 / 0
= undetermined / 0

As recall = TP/TP+FN,

recall for class 1 = 135 / (135+70+0)
= 0.65854

recall for class 2 = 108 / (70+108+0)
= 0.60674

recall for class 3 = 0 / 148+69+0
= 0

As F1 Score = 2*(Recall * Precision) / (Recall + Precision),

F1 score for class 1 = 2*(0.65854 * 0.38244) / (0.65854 + 0.38244)
= 0.48387

F1 score for class 2 = 2*(0.60674 * 0.43725) / (0.60674 + 0.43725)
= 0.50824

F1 score for class 3 = undetermined / 0

Problem2

Q1.

The prior of C_1 is 0.34208, C_2 is 0.32792 and C_3 is 0.33.

```
#Calc P(Ci)
pC1 = len(classes[1])/len(input2_train)
pC2 = len(classes[2])/len(input2_train)
pC3 = len(classes[3])/len(input2_train)
pCi = [pC1, pC2, pC3]
print("pCi value: ",pCi)
```

```
pCi value:  [0.34208333333333335, 0.3279166666666667, 0.33]
```

Q2.

m_1 = -0.02506; m_2 = 3.04947; m_3 = -2.85995;

```
#calc mean_i
mean1 = classes[1]['feature_value'].mean()
mean2 = classes[2]['feature_value'].mean()
mean3 = classes[3]['feature_value'].mean()
mean_list = [mean1, mean2, mean3]
print("mean value: ", mean_list)
```

```
mean value:  [-0.025056430115166065, 3.0494696695961974, -2.859953974797454]
```

${\sigma_1}^2$ = 1.06049; ${\sigma_2}^2$ = 3.92569; ${\sigma_3}^2$ = 9.70644;

```
#calc sd (not using std as it calc (n/n-1)*s^2)
sd1 = np.sqrt(((classes[1]['feature_value']-mean1)**2).sum()/len(classes[1]))
sd2 = np.sqrt(((classes[2]['feature_value']-mean2)**2).sum()/len(classes[2]))
sd3 = np.sqrt(((classes[3]['feature_value']-mean3)**2).sum()/len(classes[3]))
sd_list = [sd1, sd2, sd3]
print("variance value: ", list(map(lambda x: x**2, sd_list)))
```

```
variance value:  [1.0604933047619924, 3.9256875493805583, 9.706441924363155]
```

Q3.

My discriminant function is $\log {p_i}^x + \log(1-p_i)^{1-x} + \log P(C_i)$ for prior $P(C_i)$ and the log likelihood $\log {p_i}^x + \log(1-p_i)^{1-x}$.

```
def confusion_matrix():
    #pirj = predicted result i and actual result j
    p1r1, p1r2, p1r3, p2r1, p2r2, p2r3, p3r1, p3r2, p3r3 = 0,0,0,0,0,0,0,0,0
    for i in range(len(y2_test)):
        if(y2_test.iloc[i]==1):
            if(result[i]==1): p1r1+=1
            elif(result[i]==2): p2r1+=1
            else: p3r1+=1
        elif(y2_test.iloc[i]==2):
            if(result[i]==1): p1r2+=1
            elif(result[i]==2): p2r2+=1
            else: p3r2+=1
        else:
            if(result[i]==1): p1r3+=1
            elif(result[i]==2): p2r3+=1
            else: p3r3+=1
    print('\t\t\033[96mpredict\033[0;0m\t')
    print('\t\t\033[96m1\t2\t3\033[0;0m')
    print('\t\033[96m1\033[0;0m\t'+'\033[1m'+str(p1r1)+'\t'+str(p2r1)+'\t'+str(p3r1)+'\033[0;0m')
    print('\033[96mactual\t2\033[0;0m\t'+'\033[1m'+str(p1r2)+'\t'+str(p2r2)+'\t'+str(p3r2)+'\033[0;0m')
    print('\t\033[96m3\033[0;0m\t'+'\033[1m'+str(p1r3)+'\t'+str(p2r3)+'\t'+str(p3r3)+'\033[0;0m')
    return "\033[96m---------confusion_matrix---------"
print(confusion_matrix())
print("acc: ",(y2_test==result).sum()/len(y2_test))
```

```
                predict
                1       2       3
        1       194     11      15
actual  2       42      162     2
        3       37      13      124
---------confusion_matrix---------
acc:  0.8
```

Confusion matrix is as above and along the row is the number of instances in predicted class i while along the column is the the number of instances in actual class i for 1 ≤ i ≤ 3.

Q4.

Accuracy = 80%

As precision = TP/TP+FP,

precision for class 1 = 194 / (194+42+27)
= 0.73764

precision for class 2 = 162 / (11+162+13)
= 0.87097

precision for class 3 = 124 / (15+2+124)
= 0.87943

As recall = TP/TP+FN,

recall for class 1 = 194 / (194+11+15)
= 0.88182

recall for class 2 = 162 / (42+162+2)
= 0.78641

recall for class 3 = 124 / (37+13+124)
= 0.71264

As F1 Score = 2*(Recall * Precision) / (Recall + Precision),

F1 score for class 1 = 2*(0.88182 * 0.73764) / (0.88182 + 0.73764)
= 0.80331

F1 score for class 2 = 2*(0.78641* 0.87097) / (0.78641+ 0.87097)
= 0.82653
F1 score for class 3 = 2*(0. 71264* 0.87943) / (0.71264+ 0.87943)
= 0.78730

Appendix:

```
#visualize but the class size are too similar and only single feature with 2 possible value
base = len(classes[1]) + len(classes[2]) + len(classes[3])
plt.scatter(classes[1]['feature_value'], [1]*len(classes[1]), color='r',\
            s=100, alpha=0.02)
plt.scatter(classes[2]['feature_value'], [1]*len(classes[2]), color='b',\
            s=100, alpha=0.02)
plt.scatter(classes[3]['feature_value'], [1]*len(classes[3]), color='g',\
            s=100, alpha=0.02)
plt.legend(["class 1","class 2","class 3"])
plt.show()
```

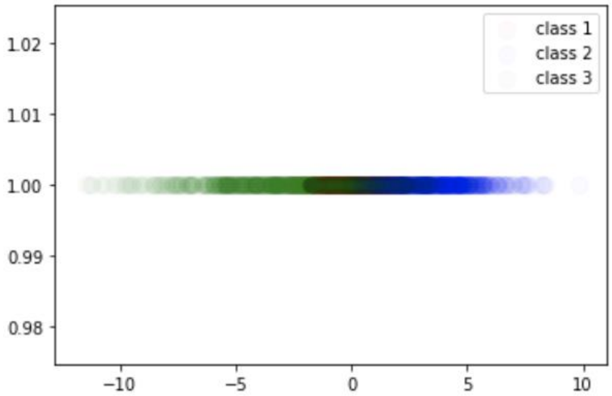

Fig.1. Visualization of Feature Distribution

```
#separate layers
base = len(classes[1]) + len(classes[2]) + len(classes[3])
plt.scatter(classes[1]['feature_value'], classes[1]['class'], color='r',
            s=len(classes[1])/(base)*100)
plt.scatter(classes[2]['feature_value'], classes[2]['class'], color='b',
            s=len(classes[2])/(base)*100)
plt.scatter(classes[3]['feature_value'], classes[3]['class'], color='g',
            s=len(classes[3])/(base)*100)
plt.legend(["class 1","class 2","class 3"])
plt.show()
```

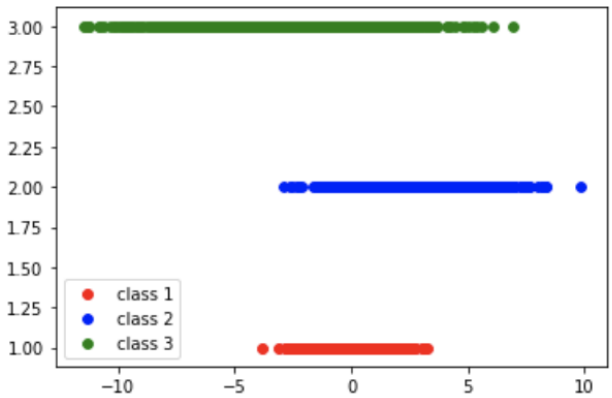

Fig.2. Visualization of Feature Distribution in Separated Layers

```
#visualization of result
#p(x|Ci)
input_pt = np.linspace(-10,10, num = 1000)
plt.plot(input_pt, stats.norm.pdf(input_pt,mean1,sd1)*pCi[0], 'r',\
         input_pt, stats.norm.pdf(input_pt,mean2,sd2)*pCi[1], 'g',\
         input_pt, stats.norm.pdf(input_pt,mean3,sd3)*pCi[2], 'b')
plt.xlim(-10,10)
plt.xlabel('x value')
plt.legend(['class1','class2','class3'])
plt.title('Approximate p(Ci|x) Distribution')
plt.show()
```

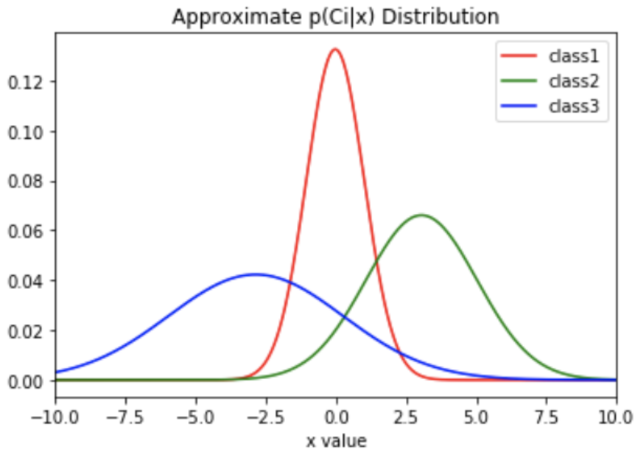

Fig.3. Visualization of Posterior Probability

```
input_pt = np.linspace(-20,20, num = 1000)
plt.plot(input_pt, g(input_pt,1), 'r',\
         input_pt, g(input_pt,2), 'g',\
         input_pt, g(input_pt,3), 'b')
plt.ylim(-20,10)
plt.xlim(-20,20)
plt.xlabel('x value')
plt.legend(['class1','class2','class3'])
plt.title('Discrimination Function')
plt.show()
```

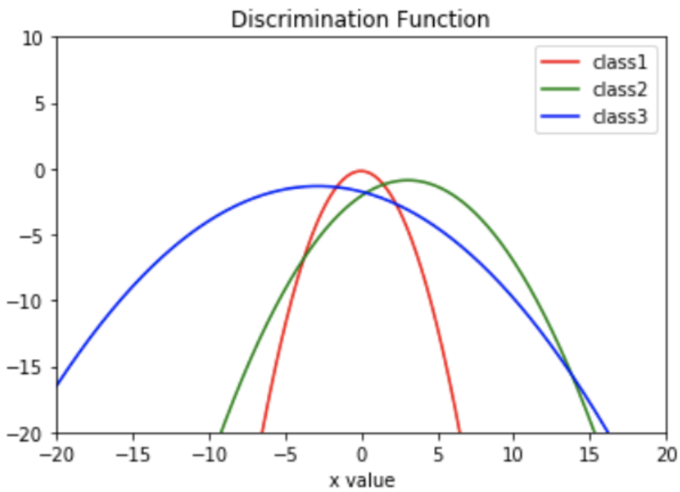

Fig.4. Visualization of Discrimination Function

Problem 3

Q1.

The prior of C_1 is 0.34125, C_2 is 0.33833 and C_3 is 0.32042.

```
pC1 = len(classes[1])/len(input3_train)
pC2 = len(classes[2])/len(input3_train)
pC3 = len(classes[3])/len(input3_train)
pCi = [pC1, pC2, pC3]
print("pCi value: ",pCi)
```

```
pCi value:  [0.32916666666666666, 0.3333333333333333, 0.3375]
```

Q2.

$p_1 = 0.1$; $p_2 = 0.515$; $p_3 = 0.77160$;

```
#calc pi
p1 = (classes[1]['feature_value_1']==1).sum()/len(classes[1])
p2 = (classes[2]['feature_value_1']==1).sum()/len(classes[2])
p3 = (classes[3]['feature_value_1']==1).sum()/len(classes[3])
pi = [p1, p2, p3]
print("pi value: ",pi)
```

```
pi value:  [0.1, 0.515, 0.7716049382716049]
```

$m_1 = 1.02467$; $m_2 = 4.96953$; $m_3 = 9.88860$;

```
#calc mean_i
mean1 = classes[1]['feature_value_2'].mean()
mean2 = classes[2]['feature_value_2'].mean()
mean3 = classes[3]['feature_value_2'].mean()
mean_list = [mean1, mean2, mean3]
print("mean value: ", mean_list)
```

```
mean value:  [1.0246692865939961, 4.969530068401736, 9.888599469860715]
```

${\sigma_1}^2 = 0.24928$; ${\sigma_2}^2 = 6.96917$; ${\sigma_3}^2 = 20.61892$;

```
#calc sd
sd1 = np.sqrt(((classes[1]['feature_value_2']-mean1)**2).sum()/len(classes[1]))
sd2 = np.sqrt(((classes[2]['feature_value_2']-mean2)**2).sum()/len(classes[2]))
sd3 = np.sqrt(((classes[3]['feature_value_2']-mean3)**2).sum()/len(classes[3]))
sd_list = [sd1, sd2, sd3]
print("variance value: ", list(map(lambda x: x**2, sd_list)))
```

```
variance value:  [0.2492844106885291, 6.969167050348478, 20.618919124235582]
```

Q3.

My discriminant function is $\log {p_i}^x + \log(1-p_i)^{1-x} + \log P(C_i)$ for prior $P(C_i)$ and the log likelihood $\log {p_i}^x + \log(1-p_i)^{1-x}$.

```
def confusion_matrix():
    #pirj = predicted result i and actual result j
    p1r1, p1r2, p1r3, p2r1, p2r2, p2r3, p3r1, p3r2, p3r3 = 0,0,0,0,0,0,0,0,0
    for i in range(len(y3_test)):
        if(y3_test.iloc[i]==1):
            if(result[i]==1): p1r1+=1
            elif(result[i]==2): p2r1+=1
            else: p3r1+=1
        elif(y3_test.iloc[i]==2):
            if(result[i]==1): p1r2+=1
            elif(result[i]==2): p2r2+=1
            else: p3r2+=1
        else:
            if(result[i]==1): p1r3+=1
            elif(result[i]==2): p2r3+=1
            else: p3r3+=1
    print('\t\t\033[96mpredict\033[0;0m\t')
    print('\t\t\033[96m1\t2\t3\033[0;0m')
    print('\t\033[96m1\033[0;0m\t'+'\033[1m'+str(p1r1)+'\t'+str(p2r1)+'\t'+str(p3r1)+'\033[0;0m')
    print('\033[96mactual\t2\033[0;0m\t'+'\033[1m'+str(p1r2)+'\t'+str(p2r2)+'\t'+str(p3r2)+'\033[0;0m')
    print('\t\033[96m3\033[0;0m\t'+'\033[1m'+str(p1r3)+'\t'+str(p2r3)+'\t'+str(p3r3)+'\033[0;0m')
    return "\033[96m---------confusion_matrix---------"
print(confusion_matrix())
print("acc: ",(y3_test==result).sum()/len(y3_test))
```

```
                predict
                1       2       3
        1       204     9       0
actual  2       15      144     27
        3       4       50      147
---------confusion_matrix---------
acc:  0.825
```

Confusion matrix is as above and along the row is the number of instances in predicted class i while along the column is the the number of instances in actual class i for 1 ≤ i ≤ 3.

Q4. accuracy = 82.5%

As precision = TP/TP+FP,

precision for class 1 = 204 / (204+15+4)
= 0.91480

precision for class 2 = 144 / (9+144+50)
= 0.70936

precision for class 3 = 147 / (0+27+147)
= 0.84483

As recall = TP/TP+FN,

recall for class 1 = 204 / (204+9+0)
= 0.95775

recall for class 2 = 144 / (15+144+27)
= 0.77419

recall for class 3 = 147 / (4+50+147)
= 0.73134

As F1 Score = 2*(Recall * Precision) / (Recall + Precision),

F1 score for class 1 = 2*(0.95775 * 0.91480) / (0.95775 + 0.91480)
= 0.93578

F1 score for class 2 = 2*(0.77419 * 0.70936) / (0.77419 + 0.70936)
= 0.74036

F1 score for class 3 = 2*(0.73134 * 0.84483) / (0.73134 + 0.84483)
= 0.78400

Appendix:

```
#visualize
from mpl_toolkits.mplot3d import Axes3D
base = len(classes[1]) + len(classes[2]) + len(classes[3])

#             s=len(classes[2])/(base)*100)
plt.scatter(classes[1]['feature_value_1'], classes[1]['feature_value_2'],\
            color='r', alpha=0.02, s=100)
plt.scatter(classes[2]['feature_value_1'], classes[2]['feature_value_2'],\
            color='b', alpha=0.02, s=100)
plt.scatter(classes[3]['feature_value_1'], classes[3]['feature_value_2'],\
            color='g', alpha=0.02, s=100)

plt.legend(["class 1","class 2","class 3"])

plt.show()
```

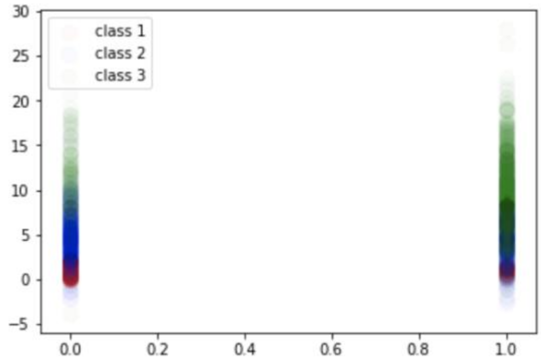

Fig.5. Visualization of Feature Distribution

```
#visualize but the class size are too similar and only single feature with 2 possible value
from mpl_toolkits.mplot3d import Axes3D
base = len(classes[1]) + len(classes[2]) + len(classes[3])
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(classes[1]['feature_value_1'],classes[1]['feature_value_2'],\
           classes[1]['class'], color='r', alpha=0.3)
ax.scatter(classes[2]['feature_value_1'], classes[2]['feature_value_2'],\
           classes[2]['class'], color='b')
ax.scatter(classes[3]['feature_value_1'], classes[3]['feature_value_2'],\
           classes[3]['class'], color='g')
ax.legend(["class 1","class 2","class 3"])
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
plt.show()
```

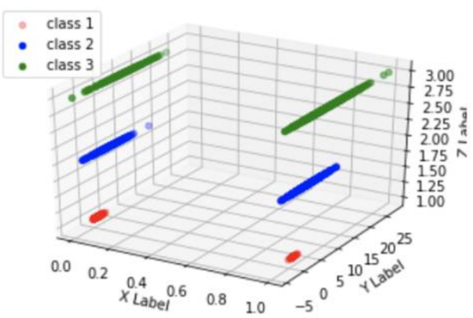

Fig.6. Visualization of Feature Distribution in Separated Layers