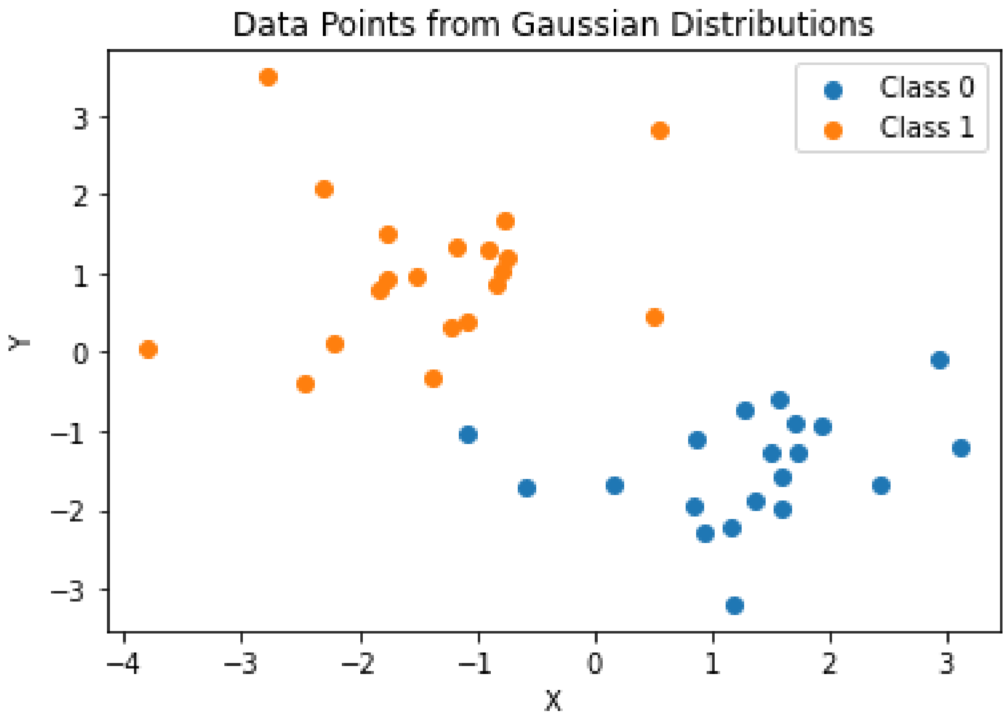
lab5

April 6, 2024

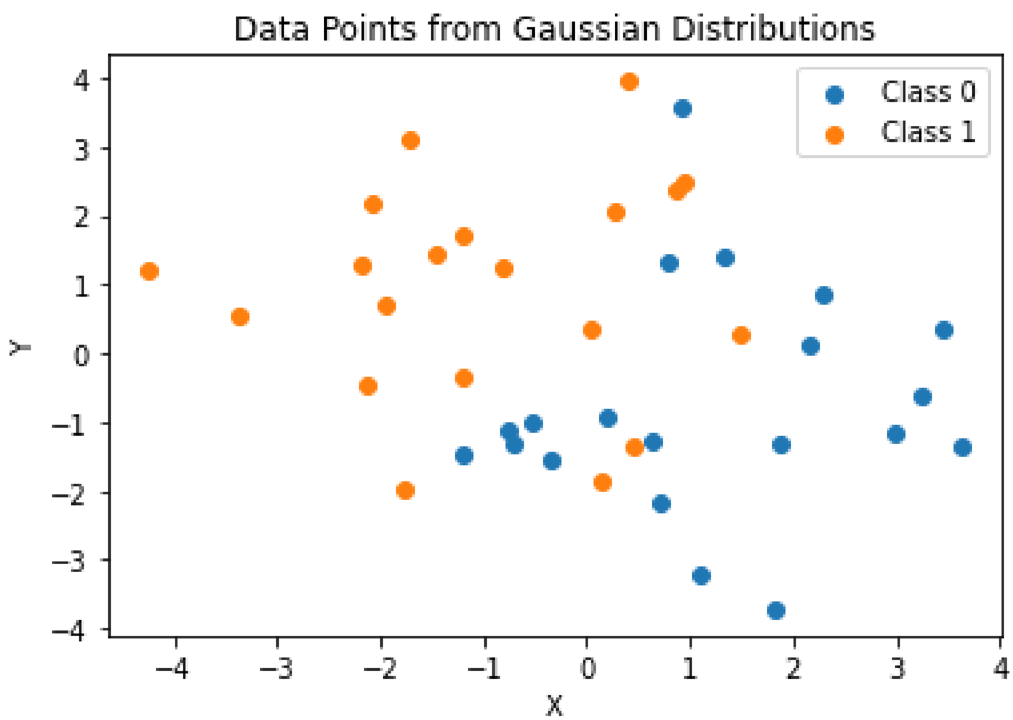
[667]:
```
#1
print("QUESTION 1")
import numpy as np
import matplotlib.pyplot as plt
import math

#variance 1
TotalNormal1=np.random.multivariate_normal(np.array([1,-1]),np.eye(2),60)
TotalNormal2=np.random.multivariate_normal(np.array([-1,1]),np.eye(2),60)

#variance=3
var31=np.random.multivariate_normal(np.array([1,-1]),np.eye(2)*3,20)
var32=np.random.multivariate_normal(np.array([-1,1]),np.eye(2)*3,20)




#different
Gauss1=TotalNormal1[:20]
Gauss2=TotalNormal2[:20]

GaussVal1=TotalNormal1[20:40]
GaussVal2=TotalNormal2[20:40]

GaussTest1=TotalNormal1[40:]
GaussTest2=TotalNormal2[40:]

def plot(Gauss01,Gauss02):
    plt.scatter(Gauss01[:, 0], Gauss01[:, 1], label='Class 0')
    plt.scatter(Gauss02[:, 0], Gauss02[:, 1], label='Class 1')
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title('Data Points from Gaussian Distributions')
    plt.legend()
    plt.show()

print("b) Given the data it appears that the classes are easily seperable. Easy␣
↪to see the difference between them and seperate via a straight line")
```

```python
plot(Gauss1,Gauss2)

print("c) Given the data it appears that the classes are not easily seperable␣
 ↪because there is a lot of overlapping  ")
plot(var31,var32)

#changing variances
mean1=np.array([1,-1])
mean2=np.array([-1,1])

variance=6
ran1=np.random.multivariate_normal(mean1,np.eye(2)*variance,20)
ran2=np.random.multivariate_normal(mean2,np.eye(2)*variance,20)

print("d) variance= "+str(variance))
print("centre 1=")
print(mean1)
print("centre 2=")
print(mean2)

plot(ran1,ran2)


mean1=np.array([2,-2])
mean2=np.array([-2,2])

variance=0.5
ran1=np.random.multivariate_normal(mean1,np.eye(2)*variance,20)
ran2=np.random.multivariate_normal(mean2,np.eye(2)*variance,20)

print(" variance= "+str(variance))
print("centre 1=")
print(mean1)
print("centre 2=")
print(mean2)

plot(ran1,ran2)

print("From the above we see that the lower the variance the more seperable\n␣
 ↪If the centres are further away from one another the more seperable")
```

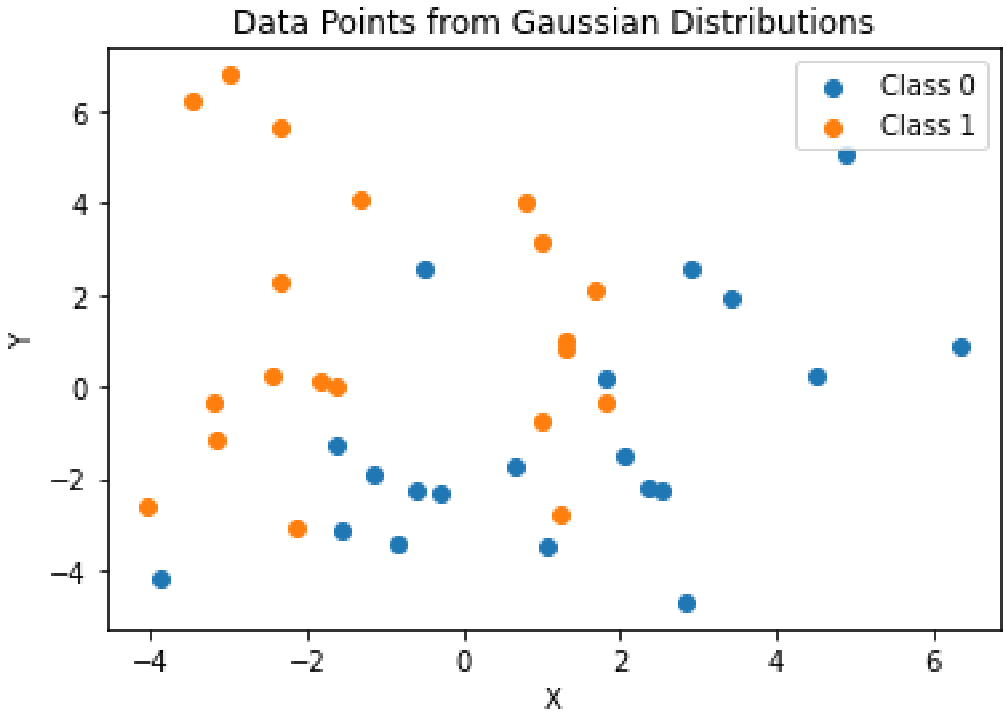
```
QUESTION 1
b) Given the data it appears that the classes are easily seperable. Easy to see
the difference between them and seperate via a straight line
```



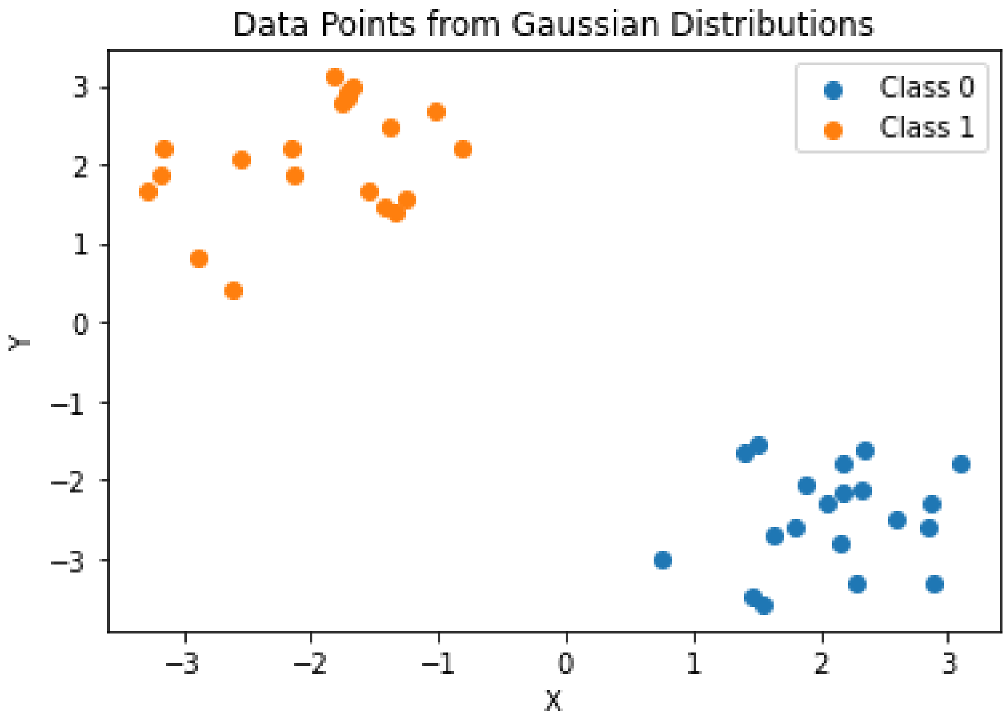
c) Given the data it appears that the classes are not easily seperable because there is a lot of overlapping



```
d) variance= 6
centre 1=
[ 1 -1]
centre 2=
[-1  1]
```




```
 variance= 0.5
centre 1=
[ 2 -2]
centre 2=
[-2  2]
```

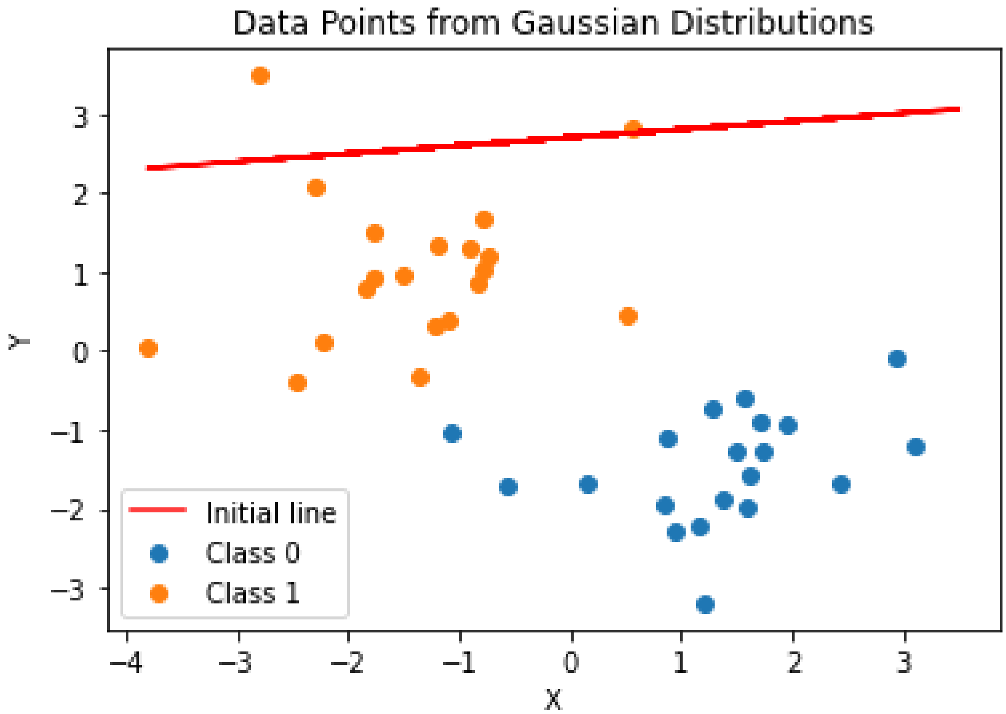



```
From the above we see that the lower the variance the more seperable
 If the centres are further away from one another the more seperable
```

QUESTION 2

[668]:
```python
#2a
import copy
theta=np.random.uniform(-0.5,0.5,[3,1])
thetaCopy= copy.deepcopy(theta)

x1_range = np.append(Gauss1,Gauss2)

x2_range = -(theta[0] + theta[1]*x1_range) / theta[2] # solving in terms of x2




print("Question 2 a")
plt.plot(x1_range, x2_range, label='Initial line', color='red')


plot(Gauss1,Gauss2)
print("Parameters:")
print(theta)
```

Question 2 a




```
Parameters:
[[ 0.48978665]
 [ 0.01861811]
 [-0.18107117]]
```

[669]:
```python
#2b
print("Question 2 b\n")

def Logistic(val):
    return 1/(1+np.exp(-val))

def h0_x(theta,Gauss):# only done for class 0
    h0=np.array([])

    for i in range(len(Gauss)):
        z= theta[0]+theta[1]*Gauss[i][0]+ theta[2]*Gauss[i][1]
        h0=np.append(h0,Logistic(z))

    return h0

import math
```

```python
def likelihood(h01, h02):
    likelihood_1 = np.log(h02).sum()
    likelihood_2 = np.log(1 - h01).sum()
    return -(likelihood_1 + likelihood_2)



print("The error is: ")
print(likelihood(h0_x(theta,Gauss1),h0_x(theta,Gauss2)))
print()

#confusion matrix , if probababilty<0.5 class 0 else class1


def confusion(Gauss,classNum,params):
    confusion = np.zeros((2,2))# rows= predicted 0 1, cols=actual 0 1
    for i in range(len(Gauss)):
        z= params[0]+params[1]*Gauss[i][0]+ params[2]*Gauss[i][1]

        if Logistic(z)<0.5:#predicted class 0
            if classNum==0:
                confusion[0][0]+=1
            else:
                confusion[0][1]+=1
        else:#preicted class is 1
            if classNum==0:
                confusion[1][0]+=1
            else:
                confusion[1][1]+=1

    return confusion


conf1=confusion(GaussTest1,0,theta)
conf2=confusion(GaussTest2,1,theta)

finalconf=np.ones((2,2))
finalconf[:,0]=conf1[:,0]
finalconf[:,1]=conf2[:,1]

print("confusion matrix:")

print(finalconf)

accuracy=(finalconf[0][0]+finalconf[1][1])/40 *100

print("accuracy is "+ str(accuracy)+"%")
```

```
Question 2 b

The error is:
34.525833640221066

confusion matrix:
[[ 0.  1.]
 [20. 19.]]
accuracy is 47.5%
```

Question 2C theta0 = theta0 – 0.01(Sigmoid (theta^T phi (x)− y)

theta1 = theta1 – 0.01(Sigmoid (theta^T phi (x)− y)(x1)

theta2 = theta2 – 0.01(Sigmoid (theta^T phi (x)− y)(x2)

[671]:
```python
#2d Gradient decent 1 itteration

# theta=np.random.uniform(-0.5,0.5,[3,1])
print("Question 2 D\n")


Old_theta=theta
print("old parameters: "+str(Old_theta))
alpha=0.01
count=0

#class 0
theta[0]=theta[0]-alpha* (h0_x(theta,GaussTest1)[0]-0)
theta[1]=theta[1]-alpha* (h0_x(theta,GaussTest1)[0]*GaussTest1[0][0])
theta[2]=theta[2]-alpha* (h0_x(theta,GaussTest1)[0]*GaussTest1[0][1])




#class1
theta[0]=theta[0]-alpha* (h0_x(theta,GaussTest2)[0]-1)
theta[1]=theta[1]-alpha* ((h0_x(theta,GaussTest2)[0]-1)*GaussTest2[0][0])
theta[2]=theta[2]-alpha* ((h0_x(theta,GaussTest2)[0]-1)*GaussTest2[0][1])





print("new paremeters: "+ str(theta))
```

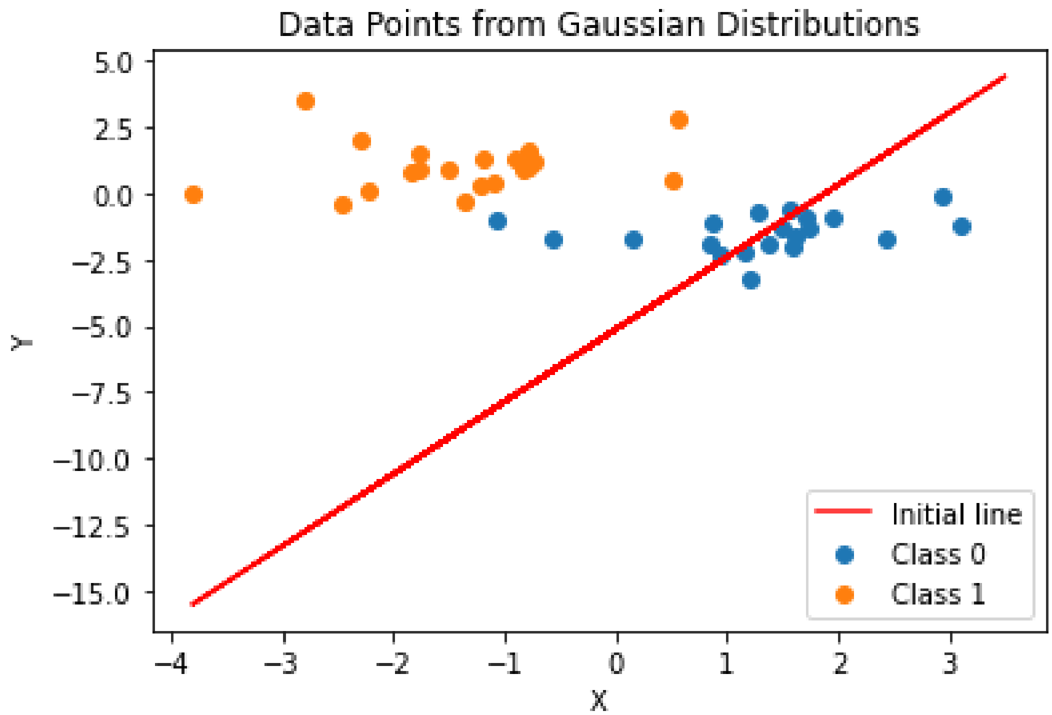
```
Question 2 D

old parameters: [[ 0.48978665]
 [ 0.01861811]
 [-0.18107117]]
```

```
new paremeters: [[ 0.48759765]
 [ 0.01057815]
 [-0.16207853]]
```

[672]:
```python
#2e Gradient decent 1 for loop



# theta=np.random.uniform(-0.5,0.5,[3,1])
print("Question 2 E\n")




Old_theta=theta

print("old parameter: "+str(Old_theta))
print("error: "+str(likelihood(h0_x(Old_theta,Gauss1),h0_x(Old_theta,Gauss2))))
print()


alpha=0.01
count=0
for i in range(len(Gauss1)):

#class 0
    theta[0]=theta[0]-alpha* (h0_x(theta,Gauss1)[i]-0)
    theta[1]=theta[1]-alpha* (h0_x(theta,Gauss1)[i]*Gauss1[i][0])
    theta[2]=theta[2]-alpha* (h0_x(theta,Gauss1)[i]*Gauss1[i][1])


    #class1
    theta[0]=theta[0]-alpha* (h0_x(theta,Gauss2)[i]-1)
    theta[1]=theta[1]-alpha* ((h0_x(theta,Gauss2)[i]-1)*Gauss2[i][0])
    theta[2]=theta[2]-alpha* ((h0_x(theta,Gauss2)[i]-1)*Gauss2[i][1])
print("new parameter: "+ str(theta))
print("error: "+str(likelihood(h0_x(theta,Gauss1),h0_x(theta,Gauss2))))


x1_range = np.append(Gauss1,Gauss2)

x2_range = -(theta[0] + theta[1]*x1_range) / theta[2] # solving in terms of x2






plt.plot(x1_range, x2_range, label='Initial line', color='red')
```

```python
plt.scatter(Gauss1[:, 0], Gauss1[:, 1], label='Class 0')
plt.scatter(Gauss2[:, 0], Gauss2[:, 1], label='Class 1')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Data Points from Gaussian Distributions')
plt.legend()
plt.show()

# conf1=confusion(Gauss1,0)
# conf2=confusion(Gauss2,1)

# finalconf=np.ones((2,2))
# finalconf[:,0]=conf1[:,0]
# finalconf[:,1]=conf2[:,1]

# print("confusion matrix")

# print(finalconf)

# accuracy=(finalconf[0][0]+finalconf[1][1])/40 *100

# print("accuracy is "+ str(accuracy)+"%")
```

```
Question 2 E

old parameter: [[ 0.48759765]
 [ 0.01057815]
 [-0.16207853]]
error: 33.70778663690564

new parameter: [[ 0.44013773]
 [-0.23422807]
 [ 0.08575809]]
error: 21.22734740523889
```



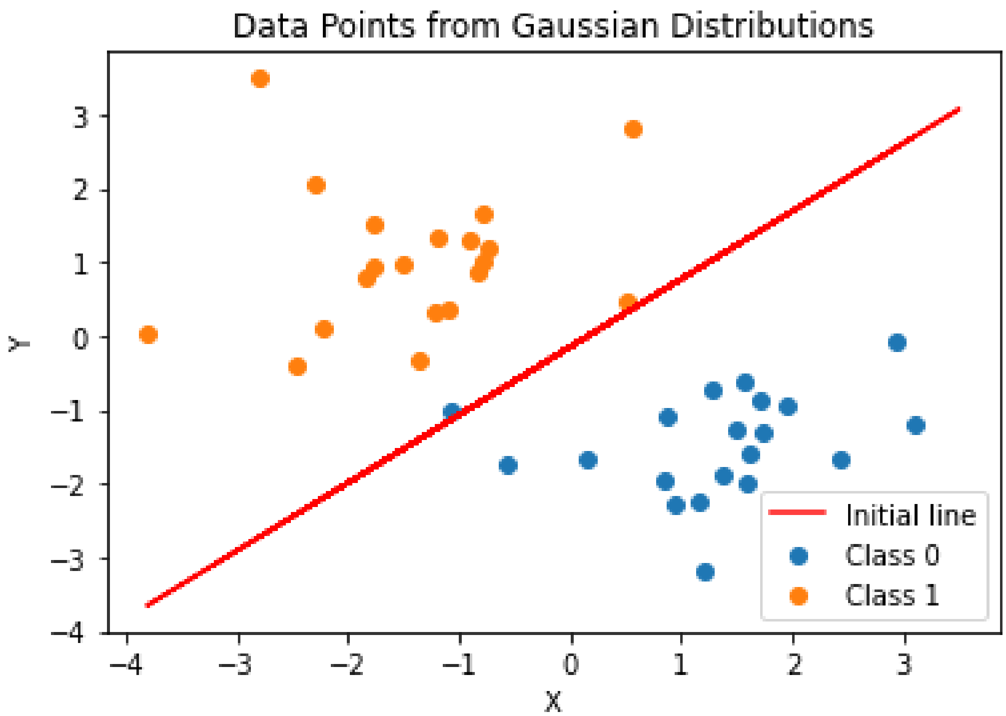
[673]:

```
#2f
import copy
print("Question 2 F\n")




# theta=np.random.uniform(-0.5,0.5,[3,1])

theta=thetaCopy
Old_theta=theta
print("old parameter: "+str(Old_theta))
print("error: "+str(likelihood(h0_x(theta,Gauss1),h0_x(theta,Gauss2))))
print()

def gradient(alpha,e,class0,class1,params):




    count=0

    temp=np.array(theta)
```

```
    while np.linalg.norm(theta-temp)<e and count<1000:

    #class 0
        temp= copy.deepcopy(theta)



        params[0]=params[0]-alpha* (h0_x(params,class0)[i]-0)
        params[1]=params[1]-alpha* (h0_x(params,class0)[i]*class0[i][0])
        params[2]=params[2]-alpha* (h0_x(params,class0)[i]*class0[i][1])


        #class1
        params[0]=params[0]-alpha* (h0_x(params,class1)[i]-1)
        params[1]=params[1]-alpha* ((h0_x(params,class1)[i]-1)*class1[i][0])
        params[2]=params[2]-alpha* ((h0_x(params,class1)[i]-1)*class1[i][1])
        count+=1

gradient(0.01,0.05,Gauss1,Gauss2,theta)



print("new parameter: "+ str(theta))
print("error: "+str(likelihood(h0_x(theta,Gauss1),h0_x(theta,Gauss2))))
x1_range = np.append(Gauss1,Gauss2)

x2_range = -(theta[0] + theta[1]*x1_range) / theta[2] # solving in terms of x2





plt.plot(x1_range, x2_range, label='Initial line', color='red')


plot(Gauss1,Gauss2)

print("The error has gone done a lot, and the decison boundary is way more␣
 ↪accurate in seperating")
```

```
Question 2 F

old parameter: [[ 0.48978665]
 [ 0.01861811]
 [-0.18107117]]
error: 34.525833640221066
```

```
new parameter: [[ 0.24463097]
 [-1.5019793 ]
 [ 1.63054928]]
error: 2.5203311724153545
```



The error has gone done a lot, and the decison boundary is way more accurate in seperating

[674]:
```
#2g testing validation data
print("Question 2 G\n")


print("training:")
print("error: "+str(likelihood(h0_x(theta,Gauss1),h0_x(theta,Gauss2))))
print()

conf1=confusion(GaussVal1,0,theta)
conf2=confusion(GaussVal2,1,theta)

print("validation: ")
print("error: "+str(likelihood(h0_x(theta,GaussVal1),h0_x(theta,GaussVal2))))
print()

finalconf=np.ones((2,2))
```

```
finalconf[:,0]=conf1[:,0]
finalconf[:,1]=conf2[:,1]

print("confusion matrix")

print(finalconf)

accuracy=(finalconf[0][0]+finalconf[1][1])/40 *100

print("accuracy is "+ str(accuracy)+"%")

print("We see that the error for the validation data is higher, this may be␣
 ↪becuase the data is unseen")
```

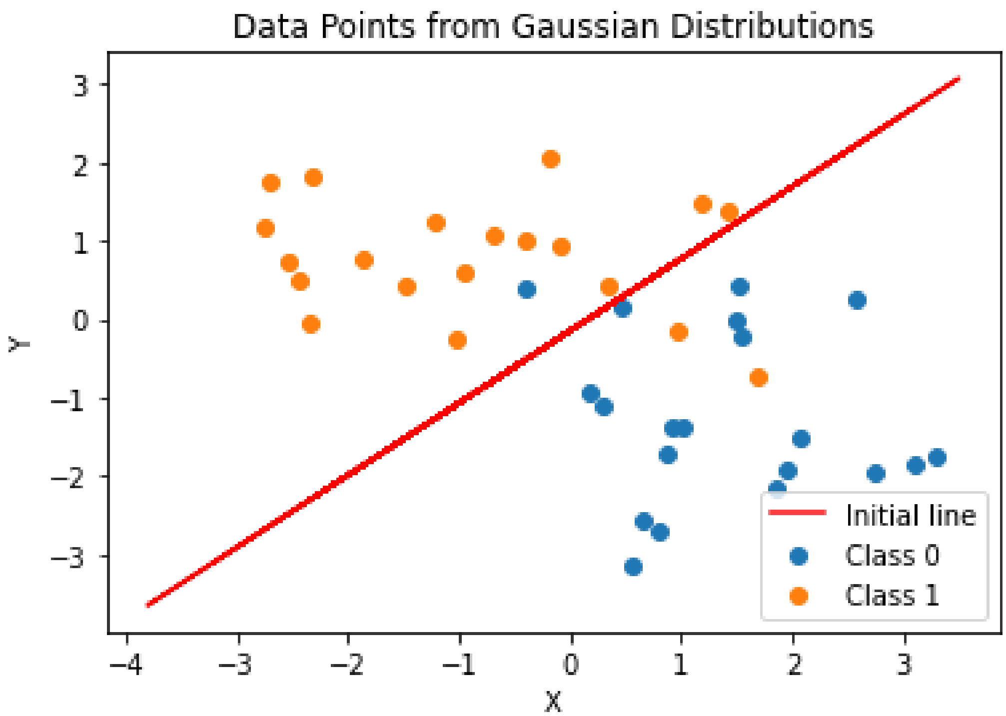
```
Question 2 G

training:
error: 2.5203311724153545

validation:
error: 10.445254767731035

confusion matrix
[[19.  2.]
 [ 1. 18.]]
accuracy is 92.5%
We see that the error for the validation data is higher, this may be becuase the
data is unseen
```

[675]:
```
print("Question 2 F\n")

import copy


theta=np.random.uniform(-0.5,0.5,[3,1])

theta=thetaCopy
Old_theta=theta
print("old parameter: "+str(Old_theta))
print("error: "+str(likelihood(h0_x(theta,Gauss1),h0_x(theta,Gauss2))))
print()

alpha=0.6
e=0.0000001
gradient(alpha,e,Gauss1,Gauss2,theta)
```

```python
print("new parameter: "+ str(theta))
print("error: "+str(likelihood(h0_x(theta,Gauss1),h0_x(theta,Gauss2))))
x1_range = np.append(Gauss1,Gauss2)

x2_range = -(theta[0] + theta[1]*x1_range) / theta[2] # solving in terms of x2





plt.plot(x1_range, x2_range, label='Initial line', color='red')


plt.scatter(GaussVal1[:, 0], GaussVal1[:, 1], label='Class 0')
plt.scatter(GaussVal2[:, 0], GaussVal2[:, 1], label='Class 1')
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Data Points from Gaussian Distributions')
plt.legend()
plt.show()


# print("training:")
# print("error: "+str(likelihood(h0_x(theta,Gauss1),h0_x(theta,Gauss2))))
# print()

conf1=confusion(GaussVal1,0,theta)
conf2=confusion(GaussVal2,1,theta)

# print("validation: ")
# print("error: "+str(likelihood(h0_x(theta,GaussVal1),h0_x(theta,GaussVal2))))
# print()

finalconf=np.ones((2,2))
finalconf[:,0]=conf1[:,0]
finalconf[:,1]=conf2[:,1]

print("confusion matrix")

print(finalconf)

accuracy=(finalconf[0][0]+finalconf[1][1])/40 *100
```

```
print("accuracy is "+ str(accuracy)+"%")


print(" From various changes we see that a higher alpha and a lower epsilon␣
↪give thee best accuracy")
```

```
Question 2 F

old parameter: [[ 0.24463097]
 [-1.5019793 ]
 [ 1.63054928]]
error: 2.5203311724153545

new parameter: [[ 0.24357221]
 [-1.52350167]
 [ 1.6533705 ]]
error: 2.4762518330105534
```




```
confusion matrix
[[19.  2.]
 [ 1. 18.]]
accuracy is 92.5%
 From various changes we see that a higher alpha and a lower epsilon give thee
```

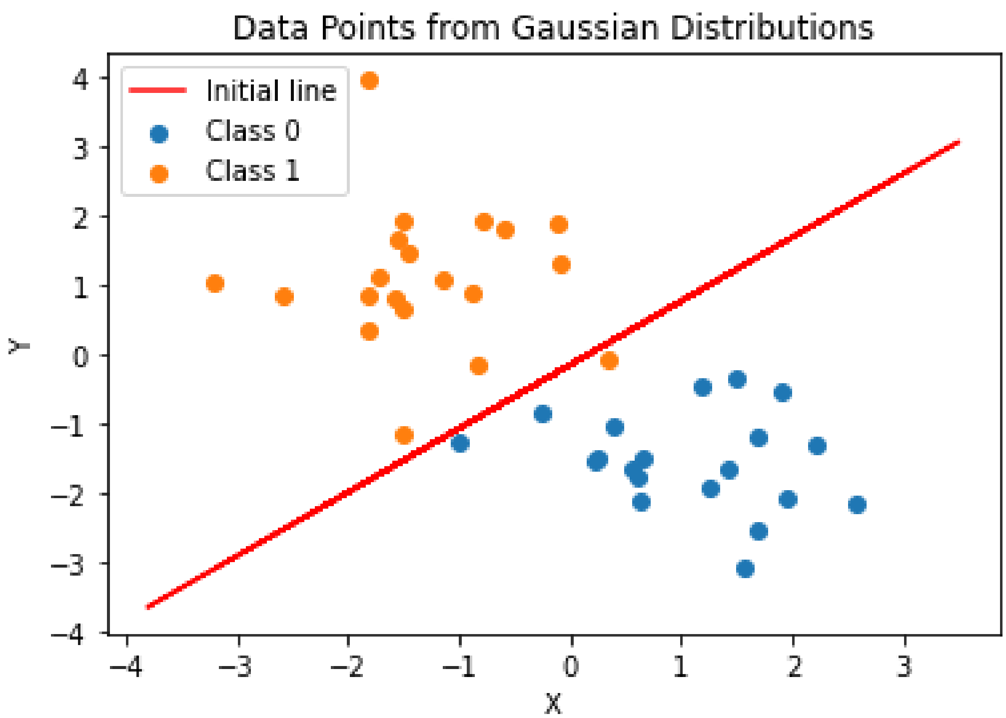
best accuracy

[676]:
```
#2i
print("Question 2 I\n")

import copy


theta=np.random.uniform(-0.5,0.5,[3,1])

theta=thetaCopy
Old_theta=theta
print("old parameter: "+str(Old_theta))
print("error: "+str(likelihood(h0_x(theta,Gauss1),h0_x(theta,Gauss2))))
print()




alpha=0.6
count=0
e=0.000001

temp=np.array(theta)

while np.linalg.norm(theta-temp)<e and count<1000:

#class 0
    temp= copy.deepcopy(theta)



    theta[0]=theta[0]-alpha* (h0_x(theta,Gauss1)[i]-0)
    theta[1]=theta[1]-alpha* (h0_x(theta,Gauss1)[i]*Gauss1[i][0])
    theta[2]=theta[2]-alpha* (h0_x(theta,Gauss1)[i]*Gauss1[i][1])


    #class1
    theta[0]=theta[0]-alpha* (h0_x(theta,Gauss2)[i]-1)
    theta[1]=theta[1]-alpha* ((h0_x(theta,Gauss2)[i]-1)*Gauss2[i][0])
    theta[2]=theta[2]-alpha* ((h0_x(theta,Gauss2)[i]-1)*Gauss2[i][1])
    count+=1



print("new parameter: "+ str(theta))
print("error: "+str(likelihood(h0_x(theta,Gauss1),h0_x(theta,Gauss2))))
```

```
x1_range = np.append(Gauss1,Gauss2)

x2_range = -(theta[0] + theta[1]*x1_range) / theta[2] # solving in terms of x2





plt.plot(x1_range, x2_range, label='Initial line', color='red')


plot(GaussTest1,GaussTest2)


print("training:")
print("error: "+str(likelihood(h0_x(theta,Gauss1),h0_x(theta,Gauss2))))
print()

conf1=confusion(GaussTest1,0,theta)
conf2=confusion(GaussTest2,1,theta)

print("testing: ")
print("error: "+str(likelihood(h0_x(theta,GaussTest1),h0_x(theta,GaussTest2))))
print()

finalconf=np.ones((2,2))
finalconf[:,0]=conf1[:,0]
finalconf[:,1]=conf2[:,1]

print("confusion matrix")

print(finalconf)

accuracy=(finalconf[0][0]+finalconf[1][1])/40 *100

print("accuracy is "+ str(accuracy)+"%")
```

```
Question 2 I

old parameter: [[ 0.24357221]
 [-1.52350167]
 [ 1.6533705 ]]
error: 2.4762518330105534

new parameter: [[ 0.2426348 ]
 [-1.54382391]
 [ 1.67486798]]
```

```
error: 2.4364461474183265
```




```
training:
error: 2.4364461474183265

testing:
error: 3.3859755860153724

confusion matrix
[[20.  1.]
 [ 0. 19.]]
accuracy is 97.5%
```

Question 2I

Training data helps in making fairly accurate predictions, therefore we use training datasets to obtain the good parameters for the desired predictions.

The validation data is used for making your model better. You do this by tweaking hyper-parameters to make the predictions more accurate.

The testing dataset is used to test the model on a previously unknown dataset to test the overall accuracy of your model