In [43]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('seaborn')
```

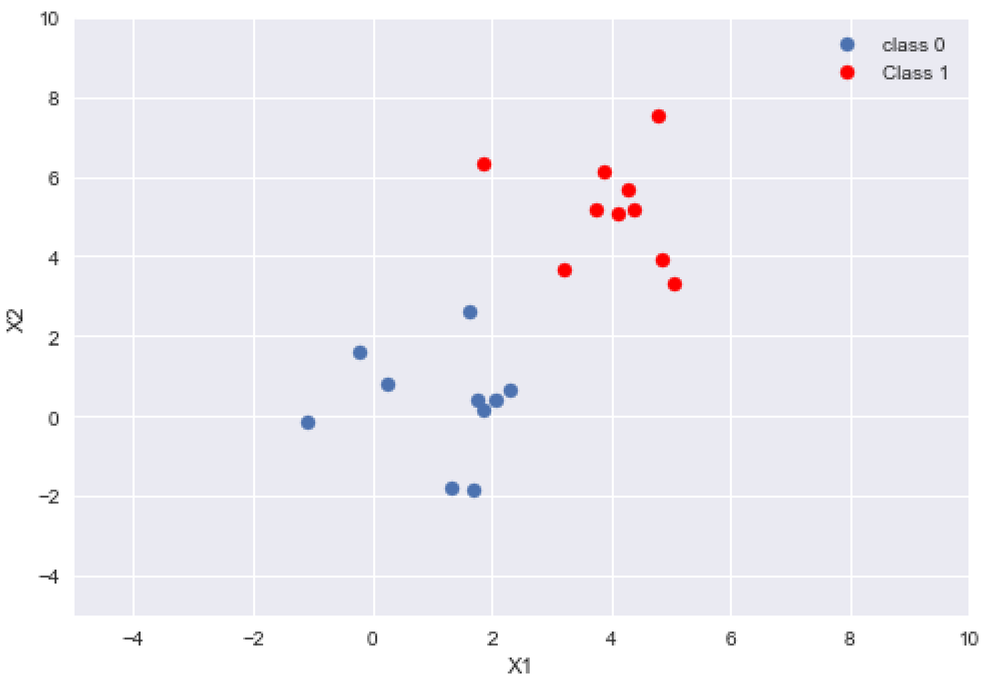
In [45]:

```
mean_01 = [1, 0.5]
cov_01 = ([[1,0.1],[0.1,1.2]])

mean_02 = [4, 5]
cov_02 = ([[1.21,0.1],[0.1,1.3]])

dist_01 = np.random.multivariate_normal(mean_01,cov_01,10)
dist_02 = np.random.multivariate_normal(mean_02,cov_02,10)
#print(dist_01)

plt.scatter(dist_01[:,0],dist_01[:,1], label='class 0')
plt.scatter(dist_02[:,0],dist_02[:,1],color='red',label='Class 1')
plt.xlim(-5,10)
plt.ylim(-5,10)
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.show()
```



In [46]:

```
# Create Training and Testing Set
#print("Dist 01", dist_01)
#print("Dist 02", dist_02)

data = np.zeros((20,3))

#print("Data", data)

data[:10,:2] = dist_01
#print("Data",data)

data[10:,:2] = dist_02
#print("Data",data)

data[10:,-1] = 1.0
print(data)
print(data.shape)
```

```
[[ 1.7641174   0.43307715  0.        ]
 [-1.10340416 -0.15589322  0.        ]
 [-0.22342016  1.6255048   0.        ]
 [ 1.32751947 -1.81703225  0.        ]
 [ 1.68789736 -1.85411563  0.        ]
 [ 1.85565222  0.14956207  0.        ]
 [ 1.6428484   2.59777231  0.        ]
 [ 0.26321739  0.78868946  0.        ]
 [ 2.07827031  0.39050626  0.        ]
 [ 2.30800632  0.65190141  0.        ]
 [ 3.22107057  3.64811157  1.        ]
 [ 5.06517669  3.31095024  1.        ]
 [ 4.27786004  5.66752066  1.        ]
 [ 4.85098393  3.93982502  1.        ]
 [ 4.38740129  5.17784477  1.        ]
 [ 4.12375282  5.08244195  1.        ]
 [ 3.8784198   6.12210224  1.        ]
 [ 4.80336076  7.53770611  1.        ]
 [ 1.85006894  6.32398468  1.        ]
 [ 3.73294603  5.15390934  1.        ]]
(20, 3)
```

In [47]:

```
np.random.shuffle(data)
print(data)
```

```
[[ 1.7641174   0.43307715  0.        ]
 [ 3.22107057  3.64811157  1.        ]
 [ 4.12375282  5.08244195  1.        ]
 [ 4.85098393  3.93982502  1.        ]
 [ 3.8784198   6.12210224  1.        ]
 [ 5.06517669  3.31095024  1.        ]
 [ 1.85565222  0.14956207  0.        ]
 [ 4.27786004  5.66752066  1.        ]
 [ 1.32751947 -1.81703225  0.        ]
 [-1.10340416 -0.15589322  0.        ]
 [ 3.73294603  5.15390934  1.        ]
 [ 0.26321739  0.78868946  0.        ]
 [-0.22342016  1.6255048   0.        ]
 [ 1.85006894  6.32398468  1.        ]
 [ 1.68789736 -1.85411563  0.        ]
 [ 4.38740129  5.17784477  1.        ]
 [ 4.80336076  7.53770611  1.        ]
 [ 1.6428484   2.59777231  0.        ]
 [ 2.30800632  0.65190141  0.        ]
 [ 2.07827031  0.39050626  0.        ]]
```

In [48]:

```
split = int(.8 * data.shape[0])
print("Split :",split)

X_train = data[:split,:-1]
X_test = data[split:,:-1]
print("X_Train",X_train)
print("X_Test",X_test)

Y_train = data[:split,-1]
Y_test = data[split:,-1]
print("Y_Train",Y_train)
print("Y_Test",Y_test)
```

```
Split : 16
X_Train [[ 1.7641174   0.43307715]
 [ 3.22107057  3.64811157]
 [ 4.12375282  5.08244195]
 [ 4.85098393  3.93982502]
 [ 3.8784198   6.12210224]
 [ 5.06517669  3.31095024]
 [ 1.85565222  0.14956207]
 [ 4.27786004  5.66752066]
 [ 1.32751947 -1.81703225]
 [-1.10340416 -0.15589322]
 [ 3.73294603  5.15390934]
 [ 0.26321739  0.78868946]
 [-0.22342016  1.6255048 ]
 [ 1.85006894  6.32398468]
 [ 1.68789736 -1.85411563]
 [ 4.38740129  5.17784477]]
X_Test [[4.80336076 7.53770611]
 [1.6428484  2.59777231]
 [2.30800632 0.65190141]
 [2.07827031 0.39050626]]
Y_Train [0. 1. 1. 1. 1. 1. 0. 1. 0. 0. 1. 0. 0. 1. 0. 1.]
Y_Test [1. 0. 0. 0.]
```

In [49]:

```
# Hypothesis - h(theta.x), old hypothesis w = theta
# x - vector (2,), eg - [5.45209039 5.36264631]
# w - (2,) for, 2 features - X1 and X2, random
# b - int
def hypothesis(x,w,b):
    h = np.dot(x,w) + b
    return sigmoid(h)

def sigmoid(z):
    return 1.0/(1.0 + np.exp(-1.0*z)) # b/w 0 & 1

def error(y_true,x,w,b):
    # x - (16,2) X_Train
    # y - (16,) Y_Train 0 or 1
    # w - (2,) for, 2 features - X1 and X2
    # b - int
    m = x.shape[0] # m- 16
    err = 0.0

    for i in range(m):
        hx = hypothesis(x[i],w,b) # b/w 0 & 1, int
        err += y_true[i] * np.log2(hx) + (1 - y_true[i])*np.log2(1-hx) # formula, int

    return -err/m #(Hence minimization, Gradient Descent), int



# caluclating gradient -
def get_grads(y_true,x,w,b):
    # x - (16,2) X_Train
    # y - (16,) Y_Train 0 or 1
    # w - (2,) for, 2 features - X1 and X2
    # b - int
    grad_w = np.zeros(w.shape) # grad_w.shape - (2,) no of grad = no of features + 1(x
    grad_b = 0.0 # int

    m = x.shape[0] # m - 16

    for i in range(m):
        hx = hypothesis(x[i],w,b) # int b/w 0&1

        grad_w += (y_true[i] - hx)*x[i] #o/p in form - (2,)
```

```
        grad_b += (y_true[i] - hx) #o/p is int

    grad_b /= m
    grad_w /= m

    return [grad_w, grad_b]



def gradient_descent(x,y_true,w,b,learning_rate=0.1):
    # x - (16,2) X_Train
    # y - (16,) Y_Train 0 or 1
    # w - (2,) for, 2 features - X1 and X2
    # b - int
    err = error(y_true,x,w,b) # int
    [grad_w, grad_b] = get_grads(y_true,x,w,b) # grad_w = (2,) , grad_b = int

    w = w + learning_rate * grad_w # w - (2,)
    b = b + learning_rate * grad_b # b - int

    return err,w,b #err- int #w - (2,) #b - int

def predict(x,w,b): # o/p - 0 or 1
    # x - (2,) eg - [5.45209039 5.36264631]
    # w - (2,) for, 2 features - X1 and X2
    # b - int
    confidence = hypothesis(x,w,b) # b/w 0 & 1
    if confidence < 0.5:
        return 0
    else:
        return 1

# On testing Data
def get_acc(x_tst, y_tst,w,b):
    # X_tst - X_Test - (4,2)
    # Y_tst - Y_Test - (4,) 0 or 1
    # w - (2,) for, 2 features - X1 and X2
    # b - int

    y_pred = [] # len - 4
    for i in range(y_tst.shape[0]): # 4
        p = predict(x_tst[i],w,b) # 0 or 1
        y_pred.append(p)
```

```
    y_pred = np.array(y_pred)

    return float((y_pred == y_tst).sum())/y_tst.shape[0] # accuracy-> 0 to 1
```

In [50]:

```
loss = [] # len - 100(after)
acc = [] # len - 100(after)

W = 2*np.random.random((X_train.shape[1],)) # W - (2,)
b = 5*np.random.random() # b - int
print(W,b)
print(X_train.shape)
```

```
[0.67192068 0.8507363 ] 3.360837732111266
(16, 2)
```

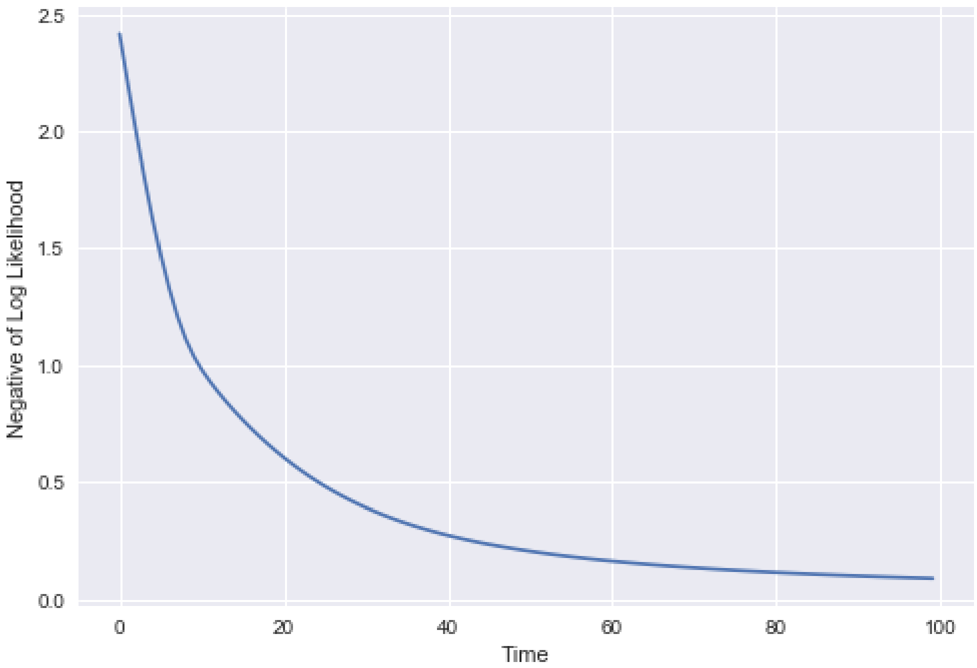
In [51]:

```
for i in range(100):
    l,W,b = gradient_descent(X_train,Y_train,W,b,learning_rate=0.5) # l -int, W-(2,), l
    acc.append(get_acc(X_test,Y_test,W,b)) # b/w 0 and 1
    loss.append(l)
print(loss)
```

```
[2.4166695811515804, 2.204318407053742, 2.0000906189094954, 1.8072634499741838, 1.6292332374169054, 1.46888133
6, 1.2092082541652385, 1.1127173984291578, 1.037683623376565, 0.978724779750845, 0.928860480285542, 0.8837297
7, 0.8020369083805352, 0.7645725737976654, 0.7290727862183618, 0.6954014071660899, 0.663455993190547, 0.63315
153, 0.5772255137698008, 0.5514791679612021, 0.5271378002245277, 0.5041459404006232, 0.482447189742351, 0.461
81506, 0.4245297377808281, 0.4074202920517874, 0.39131076998193864, 0.3761433234838493, 0.36186152153878737, 0
0619668427, 0.3237946714868738, 0.3125316280129494, 0.30190436730276793, 0.29187051741322, 0.2823903268735249
94467253111664, 0.25691219560964296, 0.24929913438370446, 0.24207758694297782, 0.23522166769914604, 0.2287073
76, 0.2166163783872767, 0.21099998572183853, 0.20564562922459168, 0.20053693390722052, 0.19565872512788948, 0.
1848145306, 0.18227136792474993, 0.17818424419281187, 0.1742667056553674, 0.1705090457500471, 0.1669022350827
16010811497046473, 0.15690567539290728, 0.15382373892896015, 0.15085594664315238, 0.14799635671912514, 0.14523
6689, 0.14001299318382612, 0.13753408267353132, 0.13513890172390963, 0.1328234309766241, 0.1305838945375695, 0
63540091176, 0.12428642860793314, 0.12231715966331727, 0.12040803534824444, 0.11855642025821436, 0.1167598261
0.11332242604299317, 0.11167729489749592, 0.11007851845445835, 0.10852421133471507, 0.10701258651714529, 0.105
4195256, 0.10271728462579896, 0.10136027924752654, 0.10003829546548I7, 0.09875002130361091, 0.0974942081767586
07526567474399, 0.09390992399I6729, 0.09277261250104138, 0.09166234876623972, 0.09057819492671176, 0.089519255
```

In [52]:

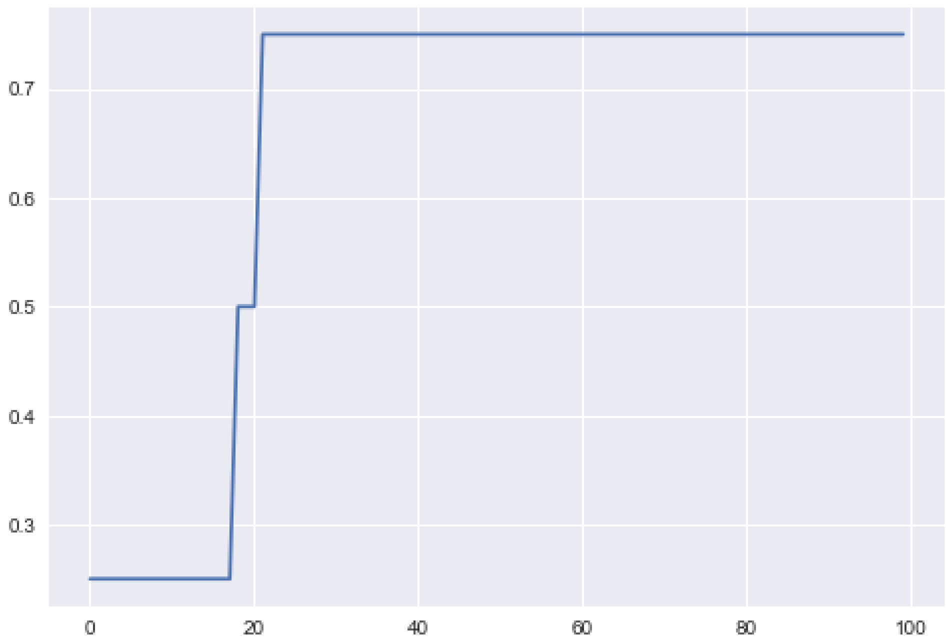
```
plt.plot(loss)
plt.ylabel("Negative of Log Likelihood")
plt.xlabel("Time")
plt.show()
```



In [53]:

```
print(acc)
plt.plot(acc)
plt.show()
print(acc[-1])
```

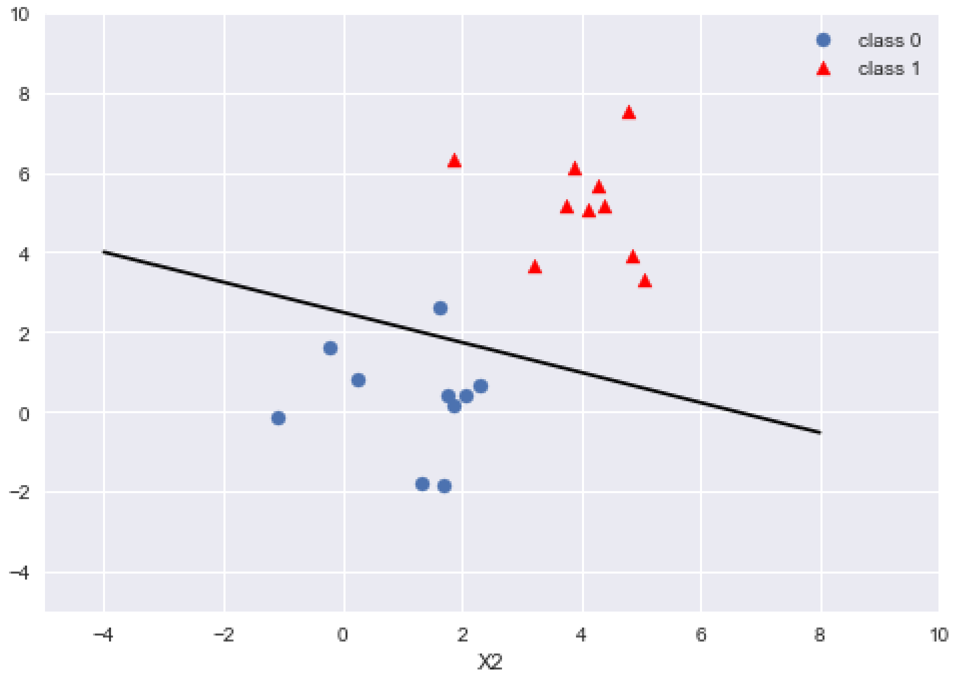
```
[0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0.25, 0
75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75
75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75
75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75
75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75, 0.75]
```




```
0.75
```

In [54]:

```
# Data Visualize
plt.scatter(dist_01[:,0],dist_01[:,1], label='class 0')
plt.scatter(dist_02[:,0],dist_02[:,1], label='class 1',color='r',marker='^')
plt.xlim(-5,10)
plt.ylim(-5,10)
plt.xlabel('X1')
plt.xlabel('X2')
x = np.linspace(-4,8,10) # 10 random point b/w -4 and 8, decimal pts.
y = -(W[0]*x + b)/W[1] # w1.x1 + w2.x2 + b = 0, y = x2, W - (2,)
plt.plot(x,y,color='k')
plt.legend()
plt.show()
```



In [55]:

```
x = np.linspace(-4,8,10)
print(x)
print(x.shape)
```

```
[-4.         -2.66666667 -1.33333333  0.          1.33333333  2.66666667
  4.          5.33333333  6.66666667  8.        ]
(10,)
```

In [ ]:

In [ ]: