In [183]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [184]:

```
plt.style.use('seaborn')
```

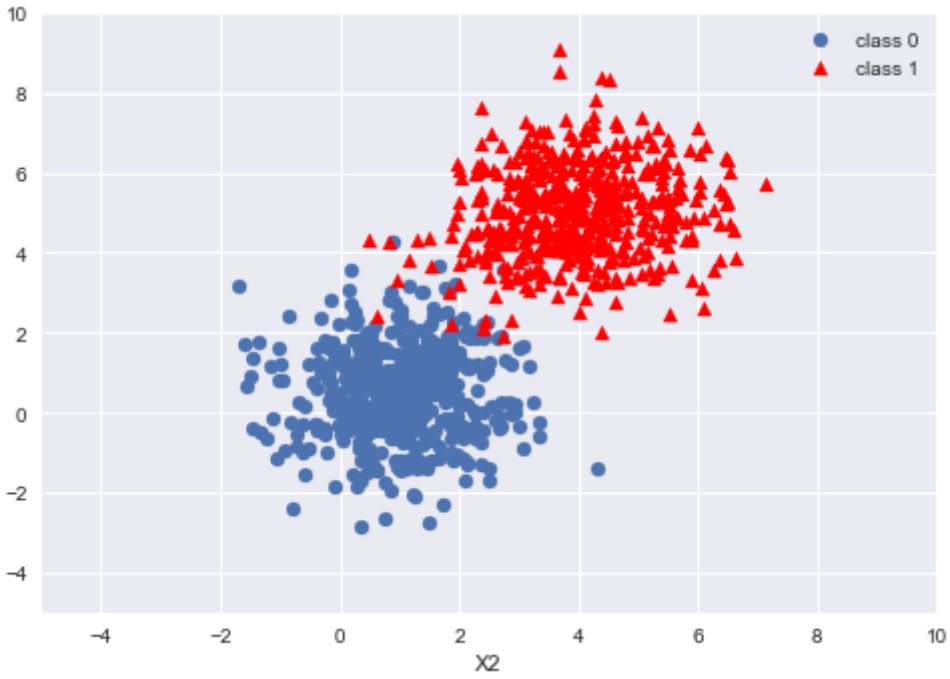
In [185]:

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

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

# Normal Distribution
dist_01 = np.random.multivariate_normal(mean_01,cov_01,500)
dist_02 = np.random.multivariate_normal(mean_02,cov_02,500)

#print(dist_01[:5,:])
#print(dist_02[:5,:])
print(dist_02.shape)
```

```
(500, 2)
```

In [186]:

```
# 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')
plt.legend()
plt.show()
```



In [187]:

```
# Create training and testing set
data = np.zeros((1000,3))
data[:500,:2] = dist_01
data[500:,:2] = dist_02
data[500:,-1] = 1.0

print(data)

np.random.shuffle(data)

split = int(.8*data.shape[0])
print("Split: ",split)
########################################
X_train = data[:split,:-1]
X_test = data[split:,:-1]

Y_train = data[:split,-1]
Y_test = data[split:,-1]

print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)
```

```
 [[0.98921353 0.07280905 0.        ]
  [0.71070403 1.49182882 0.        ]
  [0.28205739 0.5529827  0.        ]
  ...
  [4.5241158  5.3596962  1.        ]
  [2.86346341 4.42198001 1.        ]
  [5.6587323  4.86739809 1.        ]]
 Split:  800
 (800, 2) (800,)
 (200, 2) (200,)
```

In [188]:

```
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))

def error(y_true,x,w,b):
    m = x.shape[0]

    err = 0.0

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

    return -err/m

def get_grads(y_true,x,w,b):

    grad_w = np.zeros(w.shape)
    grad_b = 0.0

    m = x.shape[0]

    for i in range(m):
        hx = hypothesis(x[i],w,b)

        grad_w += (y_true[i] - hx)*x[i]
        grad_b += (y_true[i] - hx)

    grad_b /= m
    grad_w /= m

    return [grad_w, grad_b]


def gradient_descent(x,y_true,w,b,learning_rate=0.1):

    err = error(y_true,x,w,b)
    [grad_w, grad_b] = get_grads(y_true,x,w,b)
```

```
    w = w + learning_rate * grad_w
    b = b + learning_rate * grad_b

    return err,w,b

def predict(x,w,b):

    confidence = hypothesis(x,w,b)
    if confidence < 0.5:
        return 0
    else:
        return 1

def get_acc(x_tst, y_tst,w,b):
    y_pred = []
    for i in range(y_tst.shape[0]):
        p = predict(x_tst[i],w,b)
        y_pred.append(p)

    y_pred = np.array(y_pred)

    return float((y_pred == y_tst).sum())/y_tst.shape[0]
```

In [189]:

```
loss = []
acc = []

W = 2*np.random.random((X_train.shape[1],))
b = 5*np.random.random()

print(W,b)
```

```
[1.90201924 1.45540226] 4.170487718654168
```

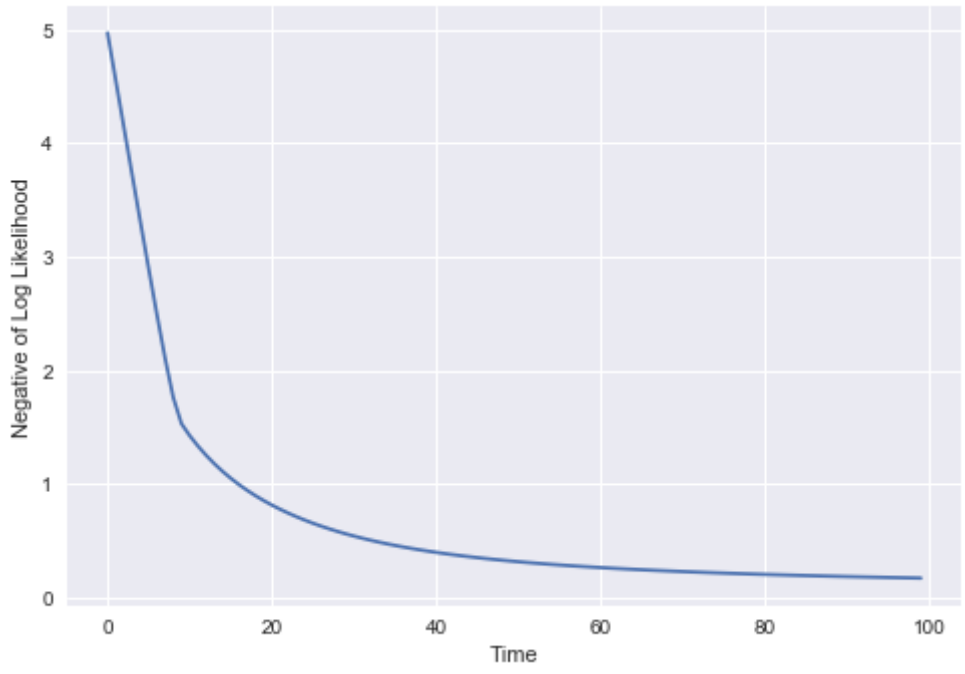
In [190]:

```
for i in range(100):
    l,W,b = gradient_descent(X_train,Y_train,W,b,learning_rate=0.5)
    acc.append(get_acc(X_test,Y_test,W,b))
    loss.append(l)
print(loss)
```

```
[4.972122014878585, 4.554240852866325, 4.137380959700796, 3.7220110039211236, 3.308951968223963, 2.899736245588
2.110224104382087, 1.7632812249129506, 1.5327272621664008, 1.4278305128298694, 1.3367354397663163, 1.254702142
1.1136082681877737, 1.0526981889153346, 0.9971628212649503, 0.9463396774460365, 0.8996640885375055, 0.85665930
7, 0.7801183100878588, 0.7459545366478183, 0.7141852396423382, 0.6845964351967845, 0.6570010872886054, 0.63123
87, 0.5846130365275927, 0.5635080160978032, 0.5437255645257307, 0.5251670372643856, 0.507742105049103, 0.49136
99, 0.46147195277322306, 0.44781518070837106, 0.43493770427164086, 0.4227842567232266, 0.41130380340210826, 0.
79705329, 0.38044687180380843, 0.3712218128488928, 0.36246751950010003, 0.35415234281428404, 0.346247052720541
15601835186505, 0.32473060097881756, 0.31821458412028125, 0.31199241605807154, 0.30604585107404786, 0.30035799
9, 0.289696956966315, 0.28469580870185207, 0.27989726421536565, 0.27528972030113336, 0.27086239075108426, 0.26
955757, 0.2585647512379257, 0.25476458939955815, 0.2511008690561404, 0.2475665152561971, 0.2441549147355408, 0
2117846555, 0.2345967084970832, 0.23161805328953392, 0.22873487917648774, 0.22594270010825027, 0.2232372994611
1807120036034294, 0.21560325050879367, 0.2132075450701176, 0.2108809553979642, 0.2086205276947983, 0.206423471
4, 0.20220906114520731, 0.2001868482991724, 0.1982182707176573, 0.19630120642831234, 0.19443364268632682, 0.19
5037446, 0.189109326153277, 0.1874215934812036, 0.18577471415860466, 0.18416720371259518, 0.18259764828090647,
670757846418, 0.17810354857500166, 0.17667294920344756, 0.17527416069189697, 0.1739061158663695]
```

In [191]:

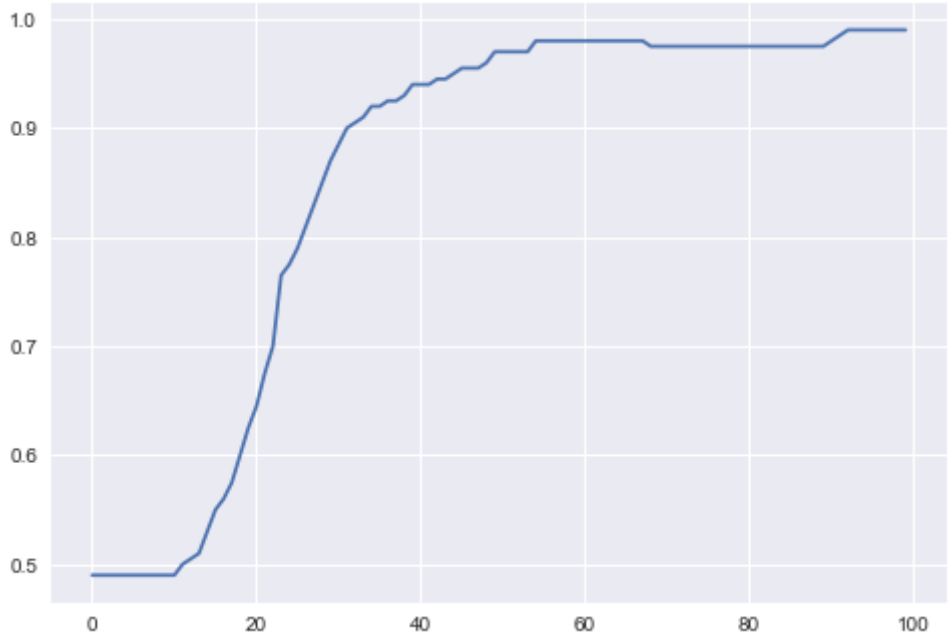
```
plt.plot(loss)
plt.ylabel("Negative of Log Likelihood")
plt.xlabel("Time")
plt.show()
```



In [192]:

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

```
[0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.49, 0.5, 0.505, 0.51, 0.53, 0.55, 0.56, 0.575, 0
7, 0.765, 0.775, 0.79, 0.81, 0.83, 0.85, 0.87, 0.885, 0.9, 0.905, 0.91, 0.92, 0.92, 0.925, 0.925, 0.93, 0.94, 0
95, 0.955, 0.955, 0.955, 0.96, 0.97, 0.97, 0.97, 0.97, 0.97, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0.98, 0
0.98, 0.975, 0.975, 0.975, 0.975, 0.975, 0.975, 0.975, 0.975, 0.975, 0.975, 0.975, 0.975, 0.975, 0.975, 0.975,
0.975, 0.975, 0.975, 0.98, 0.985, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99, 0.99]
```

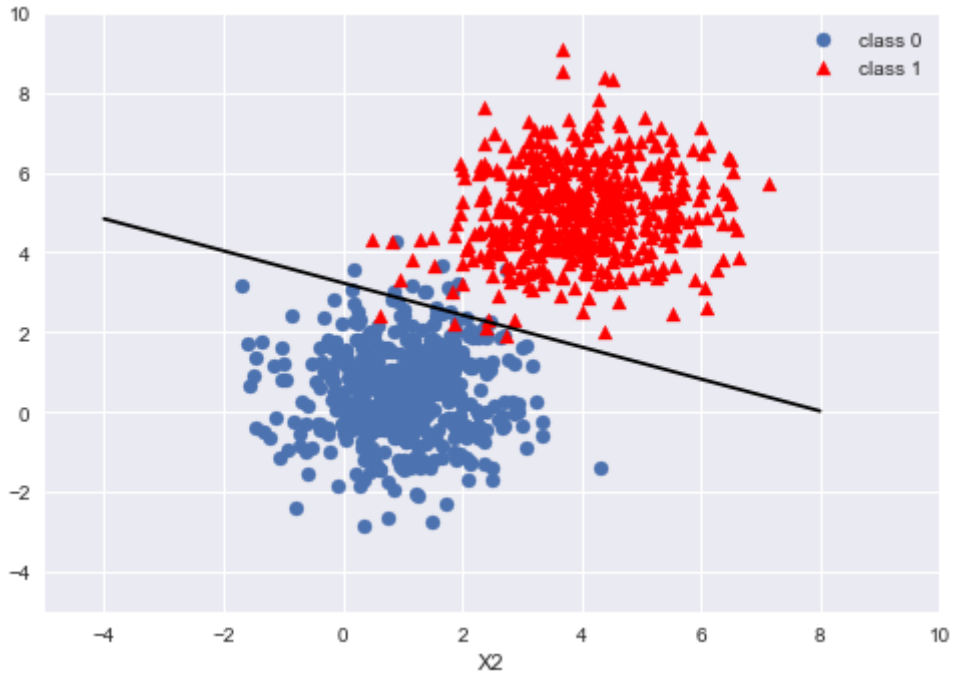



```
0.99
```

In [ ]:

In [193]:

```
# 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)
y = -(W[0]*x + b)/W[1]
plt.plot(x,y,color='k')
plt.legend()
plt.show()
```



In [ ]: