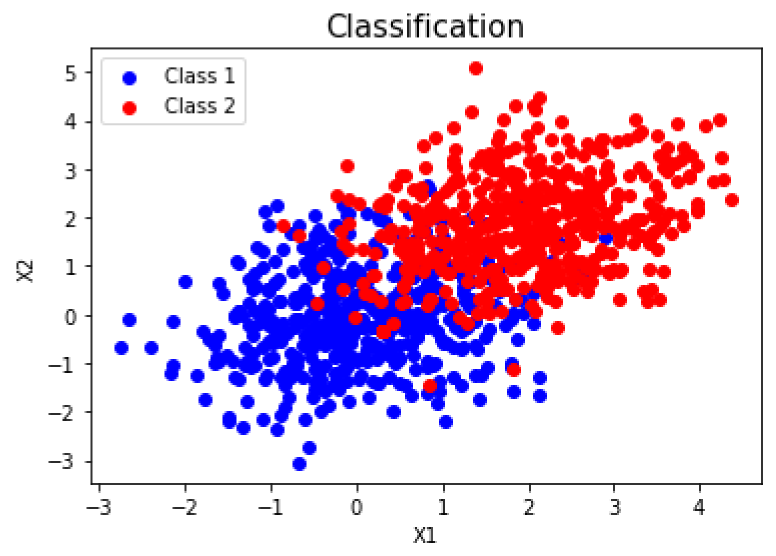
Question 4.3

In [1]:
```
import numpy as np
import math
from functools import reduce
from matplotlib import pyplot as plt
```

In [2]:
```
def multivariate_normal_pdf(x, mean, sigma):
    l = x.shape[0]
    det_S = np.linalg.det(sigma)
    norm_const = 1.0/((2.0*np.pi)**(1/2.0)*np.sqrt(det_S))
    inv_S = np.linalg.inv(sigma)
    a1 = np.dot(np.dot((x-mean), inv_S), (x-mean))

    return norm_const*np.exp(-(1.0/2.0)*a1)

def multivariate_normal_pdf_v2(x, mean, sigma):
    l = x.shape[1]
    det_S = np.linalg.det(sigma)
    norm_const = 1.0/((2.0*np.pi)**(1/2.0)*np.sqrt(det_S))
    inv_S = np.linalg.inv(sigma)
    a1    = np.sum(np.dot(x-mean, inv_S)*(x-mean), axis = 1)

    return norm_const*np.exp(-0.5*a1)
```

i.

In [3]:
```
N = 500 # Number of data points per class
S = np.array([[1, .25], [.25, 1]]) #covariance matrix
L = np.array([[0, 1], [0.005, 0]])
m1 = np.array([0, 0]) #mean for first class
m2 = np.array([2, 2]) #mean for second class

xtr1 = np.random.multivariate_normal(m1,S,N)
xtr2 = np.random.multivariate_normal(m2,S,N)


X = np.concatenate((xtr1, xtr2), axis = 0) #data_set
Y = np.concatenate((0*np.ones((N, 1)), 1*np.ones((N, 1))), axis = 0)

plt.figure(1)
plt.scatter(X[np.nonzero(Y == 0),0], X[np.nonzero(Y == 0),1], color = "b",label = "Class 1")
plt.scatter(X[np.nonzero(Y == 1),0], X[np.nonzero(Y == 1),1], color = "r",label = "Class 2")
plt.title("Classification", fontsize=15)
plt.legend(loc = 0)
plt.xlabel("X1");
plt.ylabel("X2");
```

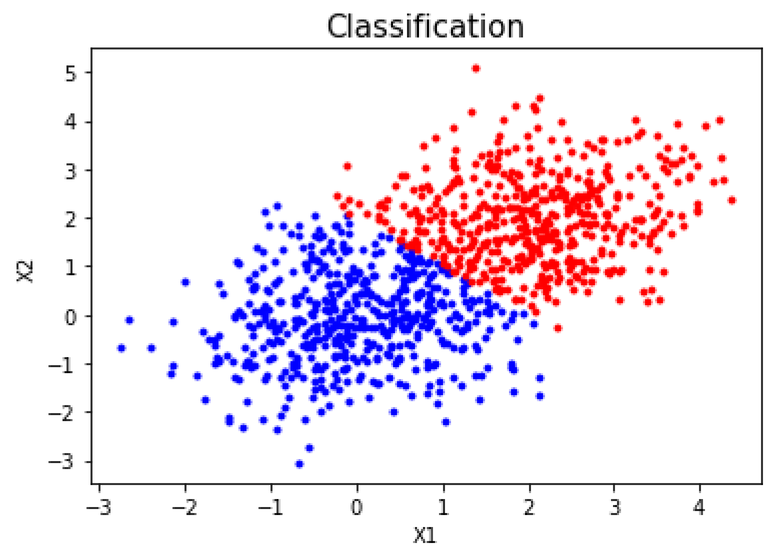


ii. & iii.

In [4]:
```
# Bayes classification of X
# Estimation of priori probabilities
p = 2*N
P2 = P1 = 0.5
p1 = np.zeros(p)
p2 = np.zeros(p)

# Estimation of pdf's for each data point
p1=multivariate_normal_pdf_v2(X,m1 ,S); # prior propability Gaussian_PDF
p2=multivariate_normal_pdf_v2(X,m2 ,S);

classf = np.zeros(p)

for i in range(0, p):
    if P1*p1[i] > P2*p2[i]:
        classf[i] = 0
    else:
        classf[i] = 1

# Error probability estimation
Pe = 0  # Probability of error
for i in range(0, p):
    if classf[i] != Y[i][0]:
        Pe += 1

Pe /= p
print('Pe: %f' % Pe)
```

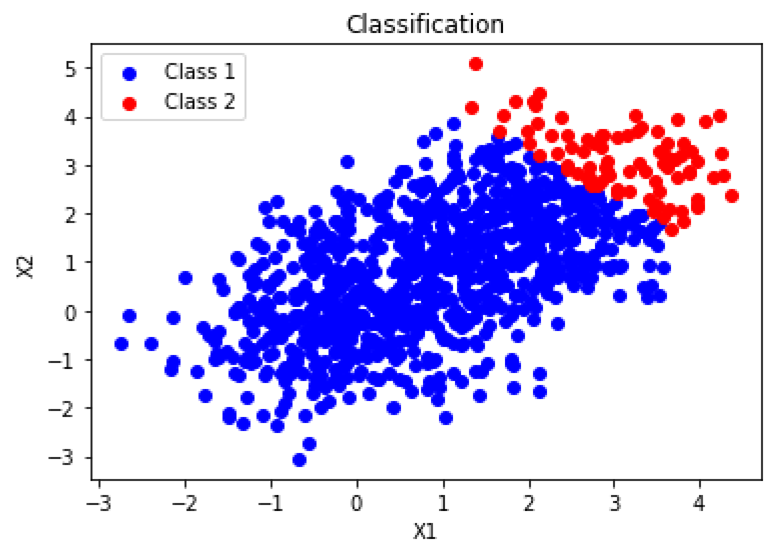
```
Pe: 0.106000
```

In [5]:
```
plt.figure(1)
plt.plot(X[np.nonzero(classf == 0),0], X[np.nonzero(classf == 0),1], '.b')
plt.plot(X[np.nonzero(classf == 1),0], X[np.nonzero(classf == 1),1], '.r')
plt.title("Classification", fontsize=15)

plt.xlabel("X1");
plt.ylabel("X2");
```



iv.

In [6]:
```
p1=multivariate_normal_pdf_v2(X,m1 ,S);
p2=multivariate_normal_pdf_v2(X,m2 ,S);

# Classification of the data points
class_loss = np.zeros(p)
for i in range(0, p):
    if L[0][1] * P1 * p1[i] > L[1][0] * P2 * p2[i]:
        class_loss[i] = 0
    else:
        class_loss[i] = 1

plt.figure(1)
plt.scatter(X[np.nonzero(class_loss == 0),0], X[np.nonzero(class_loss == 0),1], color = "b",label = "Class 1")
plt.scatter(X[np.nonzero(class_loss == 1),0], X[np.nonzero(class_loss == 1),1], color = "r",label = "Class 2")
plt.title(r"Classification", fontsize=12)
plt.legend(loc = 0)

plt.xlabel("X1");
plt.ylabel("X2");
```



v.

In [7]:
```
# Error probability estimation
Avgrisk = 0  # Average risk
for i in range(0, p):
    if class_loss[i] != Y[i][0]:
        if Y[i][0] == 0:
            Avgrisk = Avgrisk + L[0, 1]
        else:
            Avgrisk = Avgrisk + L[1, 0]
Avgrisk /= p
print('Avgrisk: %f' % Avgrisk)
```

```
Avgrisk: 0.002070
```

vi

By seen the graphs it can be said that Bayesian classification achieves results nearly to 10% error Compared to the classicBayes classification rule, the average risk minimization rule gives smaller values of average risk as well as very low daya points are categorized in class 2 this second case is due to the loss of class 2 of 0.005, so the points from this class give lower risk when missclassified. In the first scenario, the classification rule dictates for almost all the overlapping regions among both classes. This is because classification error on data steming from omega 2 is 'lesser' compared with another classification error on data derived from omega 1.