Answer 1

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn import mixture
from sklearn.model_selection import KFold
from scipy.stats import multivariate_normal
```

In [2]:
```
def compute_log_likelihood(m,data):
    '''
    m is the model
    data is samples
    '''
    return m.score(data)
```

In [3]:
```
def generate_GMM_samples(prior,number_of_samples,sig1,sig2,sig3,sig4,u1,
u2,u3,u4):
    '''
    Args:
    prior of class 1 = prior[0]
    prior of class 2 = prior[1]
    prior of class 3 = prior[2]
    prior of class 4 = 1-prior[0]-prior[1]-prior[2]

    number_of_samples

    class 1- u_1, sig_1
    class 2- u_2, sig_2
    class 3- u_3, sig_3
    class 4- u_4, sig_4

    x is samples from zero-mean identity-covariance Gaussian sample gene
rators

    generating class 1- A1*x+b1
    generating class 2- A2*x+b2
    generating class 3- A3*x+b3
    generating class 4- A4*x+b4

    '''
    from matplotlib.pyplot import figure
    #txt="Plot of data sampled from 4 gaussians "
    fig = plt.figure(figsize=(10,10));
    #fig.text(.35,0.05,txt,fontsize=15);

    samples_class1=[]
    samples_class2=[]
    samples_class3=[]
    samples_class4=[]

    sig_1=np.matrix(sig1)
    sig_2=np.matrix(sig2)
    sig_3=np.matrix(sig3)
    sig_4=np.matrix(sig4)

    u_1=np.matrix(u1).transpose()
    u_2=np.matrix(u2).transpose()
    u_3=np.matrix(u3).transpose()
    u_4=np.matrix(u4).transpose()

    prior=prior
    A1=np.linalg.cholesky(sig_1)
    b1=u_1

    A2=np.linalg.cholesky(sig_2)
    b2=u_2

    A3=np.linalg.cholesky(sig_3)
    b3=u_3

    A4=np.linalg.cholesky(sig_4)
```

```
    b4=u_4

    zero_mean=[0,0]
    cov=[[1,0],[0,1]]

    for i in range(number_of_samples):
        uniform_sample=np.random.uniform()

        sample_from_zero_mean_identity_covariance=np.random.multivariate
_normal(zero_mean,cov,[1]).transpose()

        if uniform_sample<prior[0]:
            '''sample from class class 1'''
            sample=A1.dot(sample_from_zero_mean_identity_covariance)+b1
            samples_class1.append(sample)
        elif (prior[0]<uniform_sample<prior[0]+prior[1]):
            '''sample from class class 2'''
            sample=A2.dot(sample_from_zero_mean_identity_covariance)+b2
            samples_class2.append(sample)
        elif (prior[0]+prior[1]<uniform_sample<prior[0]+prior[1]+prior[2
]):
            '''sample from class class 3'''
            sample=A3.dot(sample_from_zero_mean_identity_covariance)+b3
            samples_class3.append(sample)
        else :
            '''sample from class class 4'''
            sample=A4.dot(sample_from_zero_mean_identity_covariance)+b4
            samples_class4.append(sample)


    samples_class1_final=np.hstack(samples_class1)
    samples_class2_final=np.hstack(samples_class2)
    samples_class3_final=np.hstack(samples_class3)
    samples_class4_final=np.hstack(samples_class4)

    a=np.squeeze(np.asarray(samples_class1_final.transpose()[:,1]))
    b=np.squeeze(np.asarray(samples_class1_final.transpose()[:,0]))

    c=np.squeeze(np.asarray(samples_class2_final.transpose()[:,1]))
    d=np.squeeze(np.asarray(samples_class2_final.transpose()[:,0]))

    e=np.squeeze(np.asarray(samples_class3_final.transpose()[:,1]))
    f=np.squeeze(np.asarray(samples_class3_final.transpose()[:,0]))

    g=np.squeeze(np.asarray(samples_class4_final.transpose()[:,1]))
    h=np.squeeze(np.asarray(samples_class4_final.transpose()[:,0]))
    plt.xlabel('Variable x1',size=13)
    plt.ylabel('Variable x2',size=13)
    fig.suptitle('Plot of data sampled from 4 gaussians', fontsize=15)
    plt.subplots_adjust(top=.95)
    plt.scatter(b,a,color='r',marker='*',label='class 1',s=50)
    plt.scatter(d,c,color='g',marker='*',label='class 2',s=50)
    plt.scatter(f,e,color='b',marker='*',label='class 3',s=50)
    plt.scatter(h,g,color='y',marker='*',label='class 4',s=50)
    plt.legend(loc='best')
    #plt.show()
```

```
    data=np.hstack([samples_class1_final,samples_class2_final,samples_cl
ass3_final,samples_class4_final])
    '''
    data.shape=(2,samples)
    '''
    return data
```

In [4]:
```
def gmm_em_kfold(data):
    '''
    create 8 models of GMM
    '''
    n_components = np.arange(1, 9)
    models = [mixture.GaussianMixture(n, covariance_type='full', random_
state=0)
              for n in n_components]
    '''
    Lists log_likelihood,bic,aic to store result and plot
    '''
    log_likelihood=[]
    bic=[]
    aic=[]
    '''
    KFold function from sklearn
    '''
    cv = KFold(n_splits=10, random_state=42, shuffle=True)
    for m in models:
        '''
        Lists scores,scores_bic,scores_aic to store 'k' scores on 'k' di
fferent validation set
        '''
        scores = []
        scores_bic=[]
        scores_aic=[]
        for train_index, test_index in cv.split(data.T):
            X_train, X_test= data.T[train_index], data.T[test_index]
            m.fit(X_train) # fit the model
            scores.append(compute_log_likelihood(m,X_test)) # find  log_
likelihood on test set and add it to the list
            scores_bic.append(m.bic(X_test)) # find  bic on test set and
add it to the list
            scores_aic.append(m.aic(X_test)) # find  aic on test set and
add it to the list
        log_likelihood.append(-1*sum(scores)/len(scores)) # average the
 score over k validation sets
        bic.append(sum(scores_bic)/len(scores_bic)) # average the score
 over k validation sets
        aic.append(sum(scores_aic)/len(scores_aic)) # average the score
 over k validation sets
    from matplotlib.pyplot import figure
    fig = plt.figure(figsize=(10,10));
    plt.plot(n_components, log_likelihood, label='negative log_likelihoo
d')
    plt.xlabel('n_components',size=13);
    plt.ylabel('score',size=13);
    plt.legend(loc='best')
    fig.suptitle('Negative log_likelihood score for different GMM model
s', fontsize=15)
    plt.subplots_adjust(top=.95)
    plt.show()
    from matplotlib.pyplot import figure
    fig = plt.figure(figsize=(10,10));
    plt.xlabel('n_components',size=13);
    plt.ylabel('score',size=13);
```

```
    plt.plot(n_components, bic, label='BIC')
    plt.plot(n_components, aic, label='AIC')
    fig.suptitle('BIC and AIC scores for different GMM models', fontsize
=15)
    plt.subplots_adjust(top=.95)
    plt.legend(loc='best')
    plt.show()
    return
```

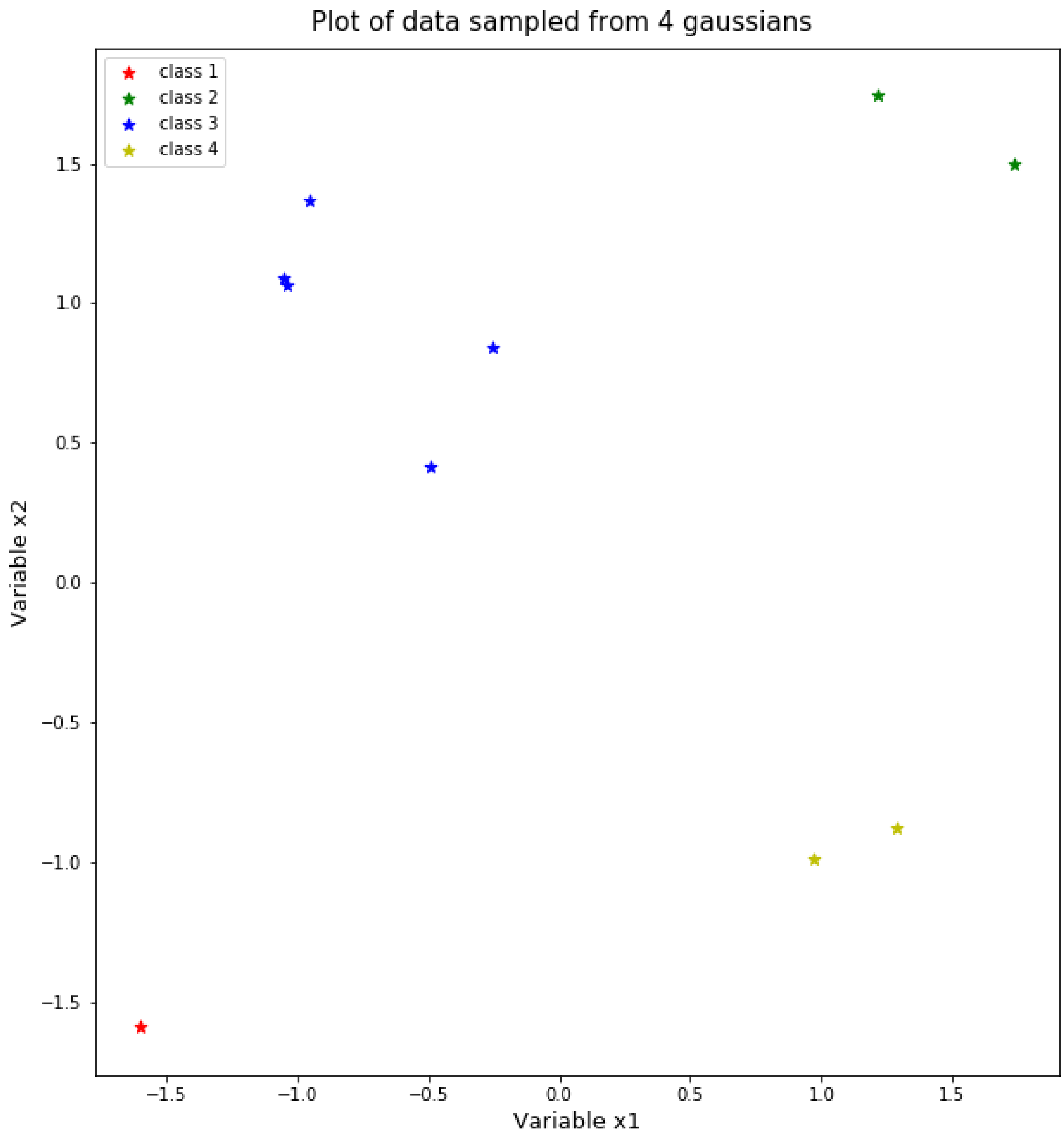
Answer part 1. True GMM

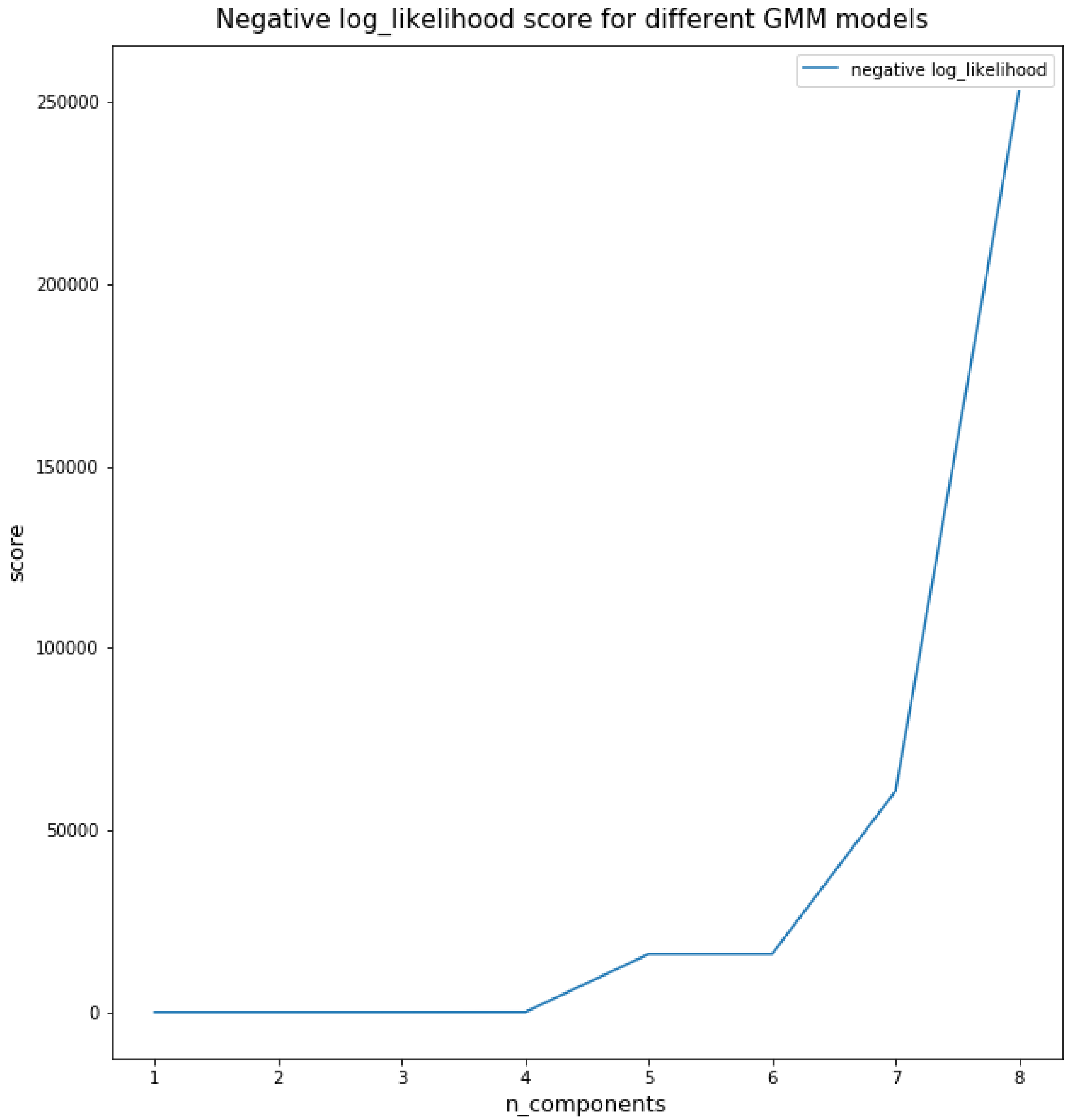
In [5]:

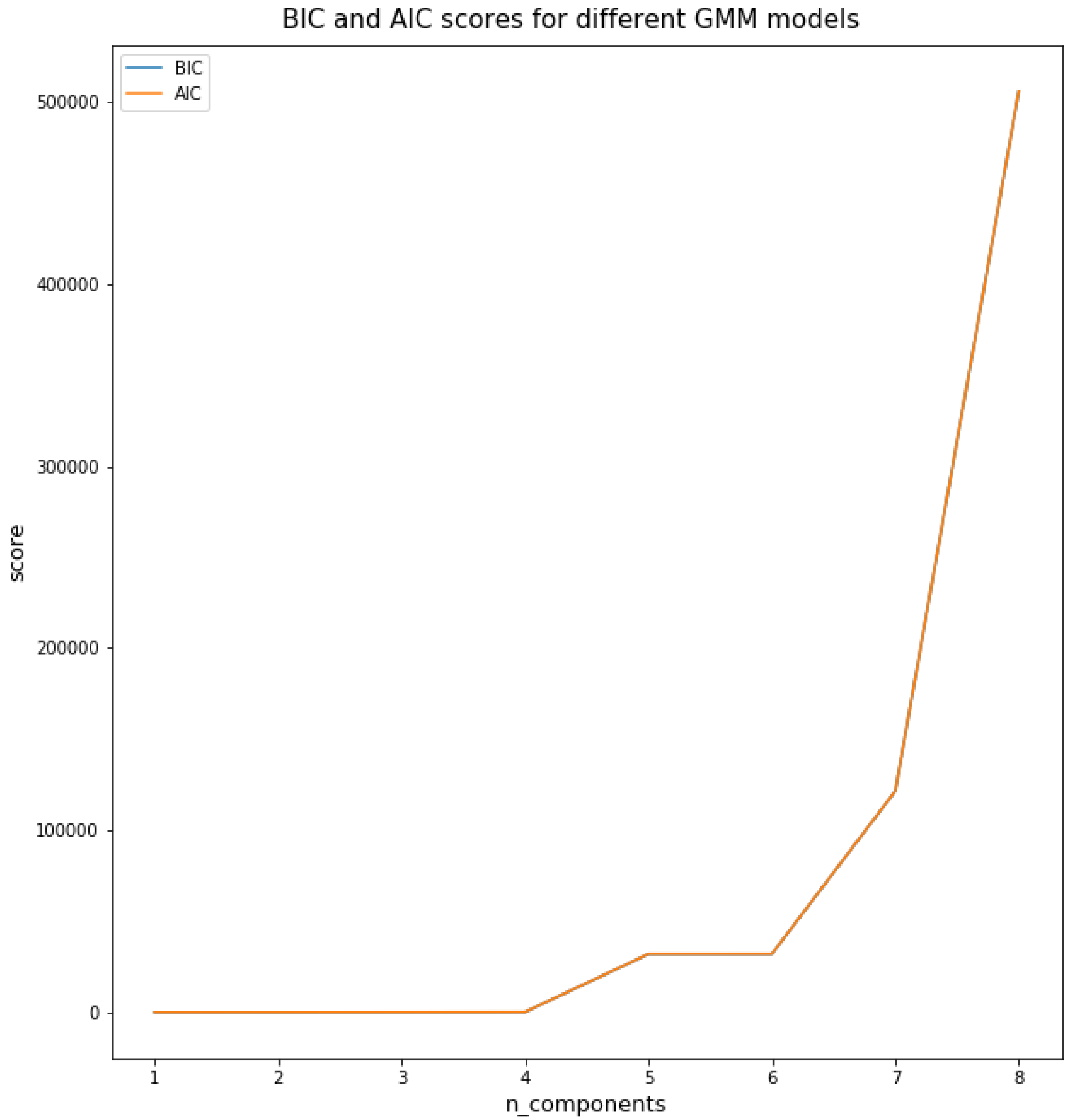
```
prior=[0.19,0.21,0.36,0.24]
# prior of Gaussian 1 = prior[0]
# prior of Gaussian 2 = prior[1]
# prior of Gaussian 3 = prior[2]
# prior of Gaussian 4 = prior[3]

# mean and covariance of gaussian 1
sig1=[[.15,-.1],[.1,.15]]
u1=[-1,-1]

# mean and covariance of gaussian 2
sig2=[[.15,.1],[.1,.15]]
u2=[1,1]

# mean and covariance of gaussian 3
sig3=[[.15,.1],[-.1,.15]]
u3=[-1,1]

# mean and covariance of gaussian 4
sig4=[[.15,.1],[-.1,.15]]
u4=[1,-1]
```

In [6]:
```
for i in [10,100, 1000, 10000]:
    print ("Number of samples: ",i)
    data=generate_GMM_samples(prior=prior,number_of_samples=i,sig1=sig1,
\
                     sig2=sig2,sig3=sig3,sig4=sig4,u1=u1,u2=u2,u3=u3,u4=
u4);
    gmm_em_kfold(data)
```

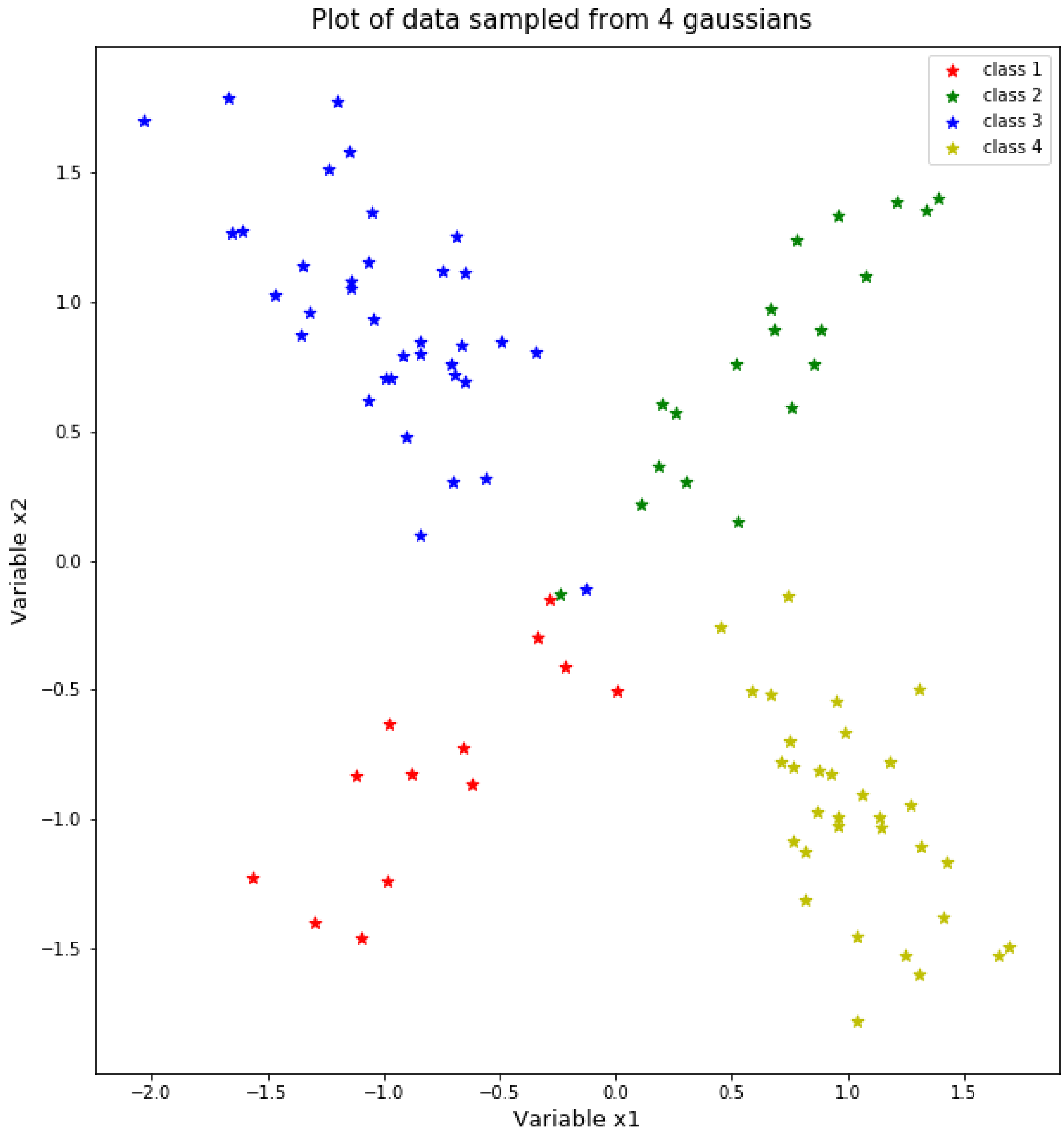
```
Number of samples:  10
```

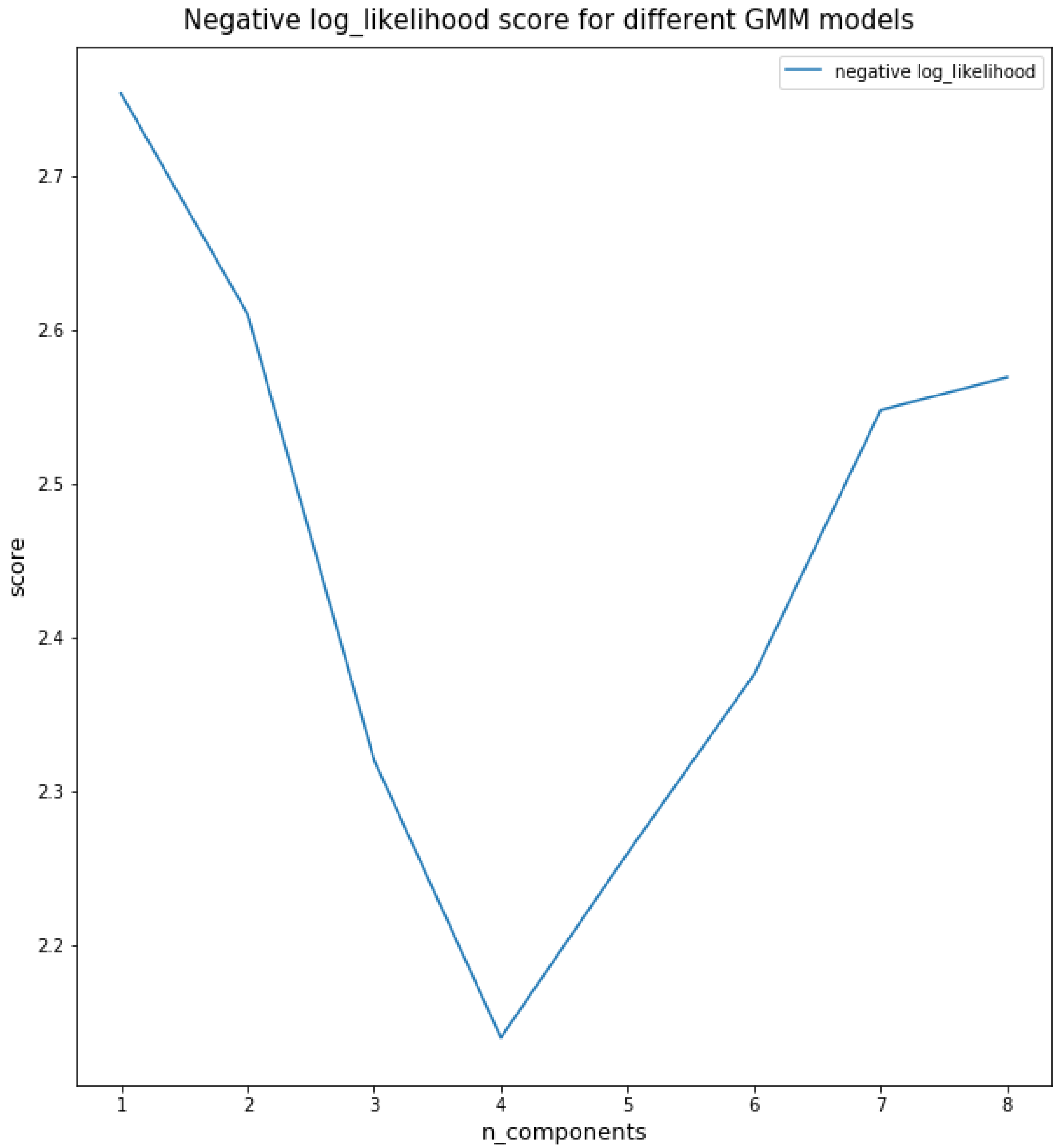


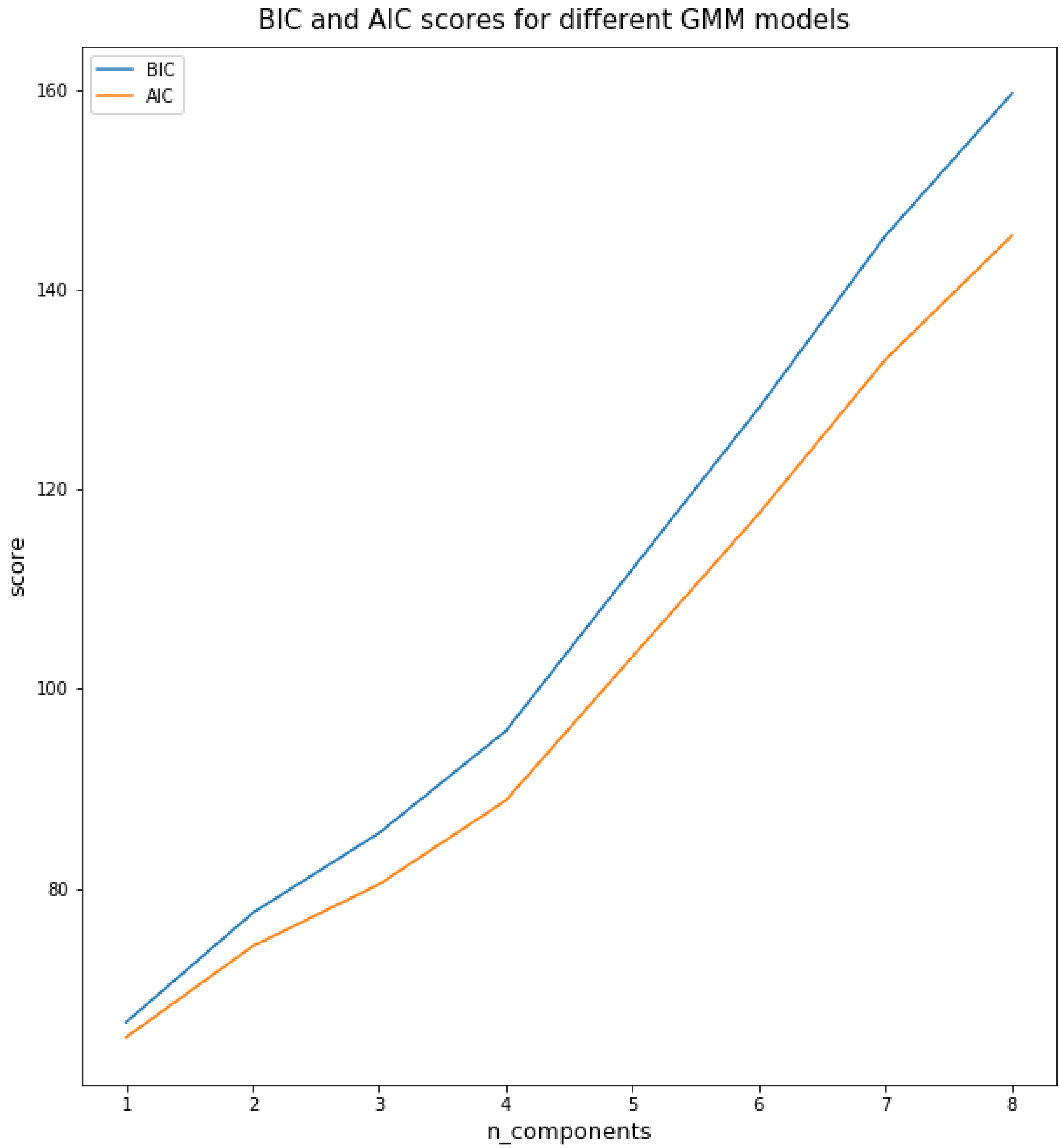

Negative log_likelihood score for different GMM models
negative log_likelihood
250000
200000
150000
100000
50000
0
score
1
2
3
4
5
6
7
8
n_components



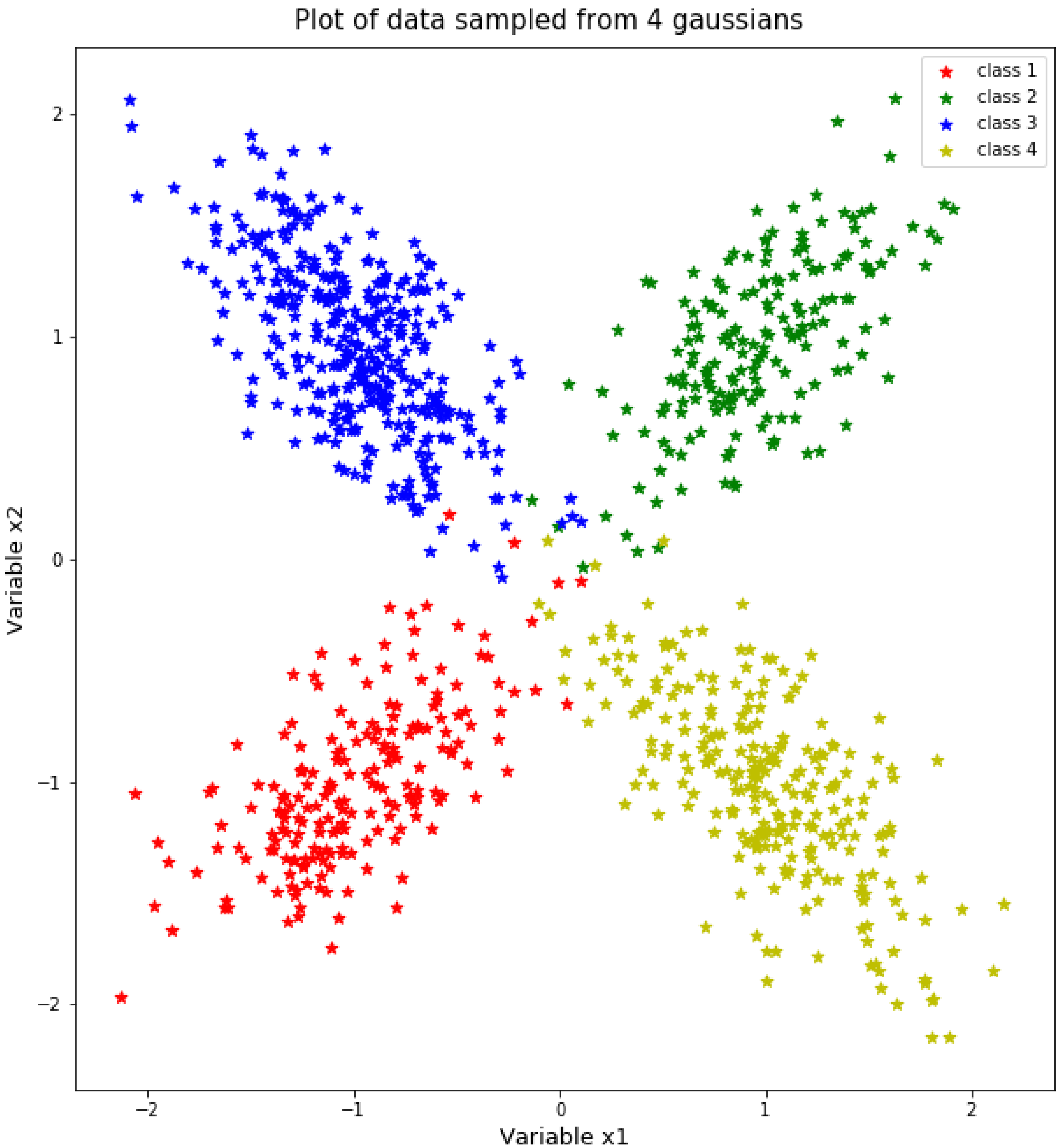

```
Number of samples:  100
```

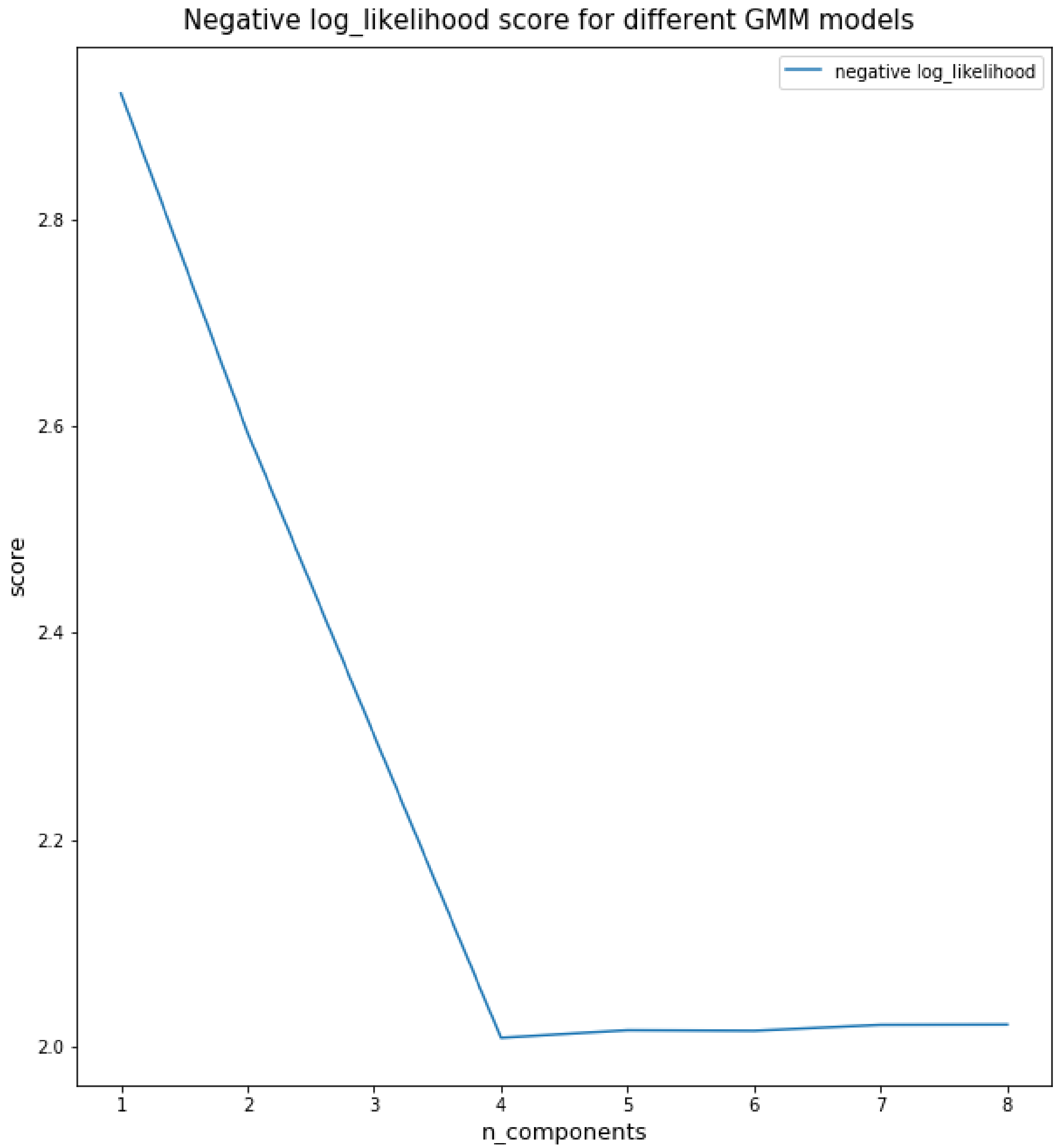

Plot of data sampled from 4 gaussians
class 1
class 2
class 3
class 4
Variable x2
Variable x1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

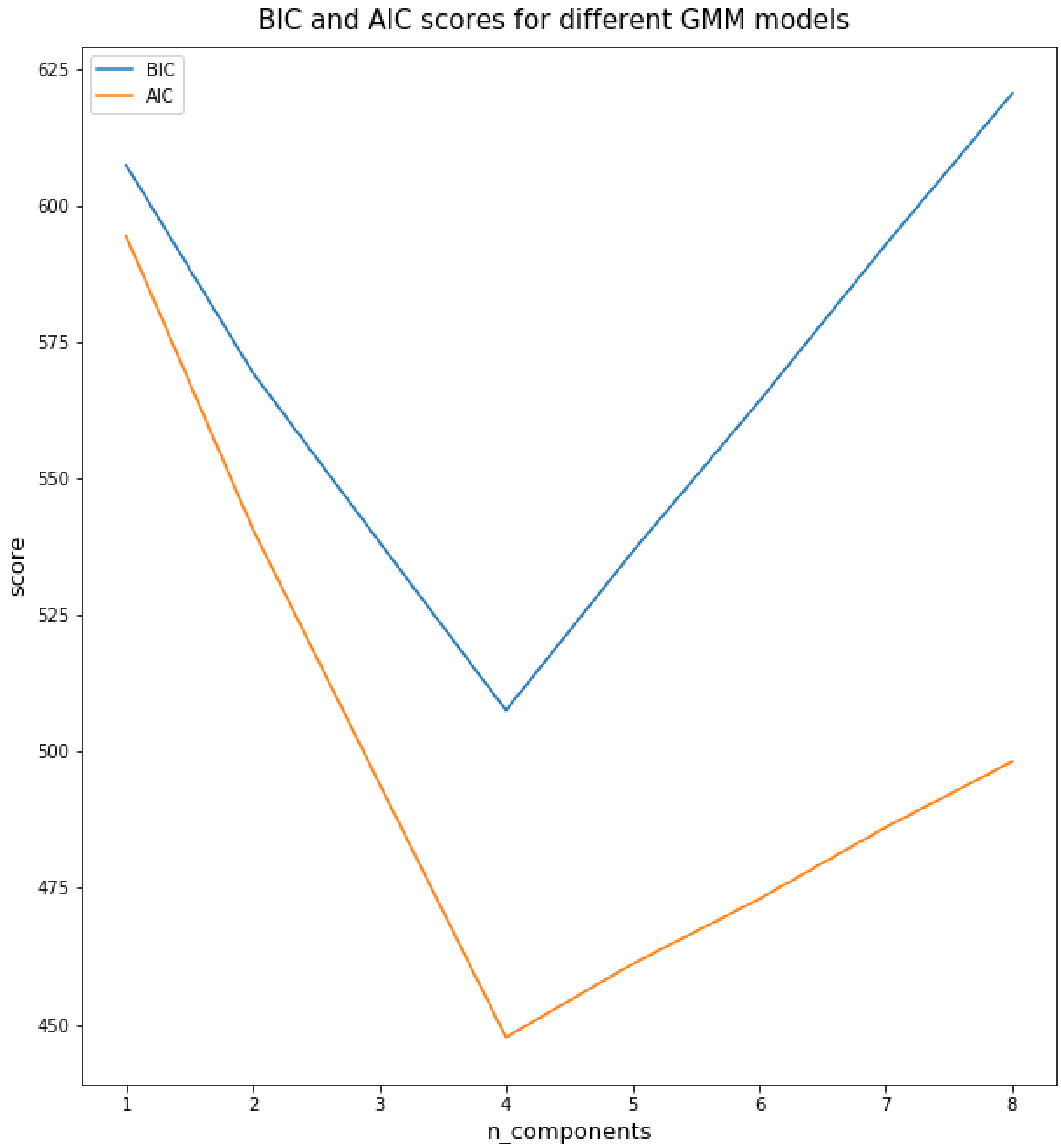

Negative log_likelihood score for different GMM models
negative log_likelihood
score
n_components
2.7
2.6
2.5
2.4
2.3
2.2
1
2
3
4
5
6
7
8



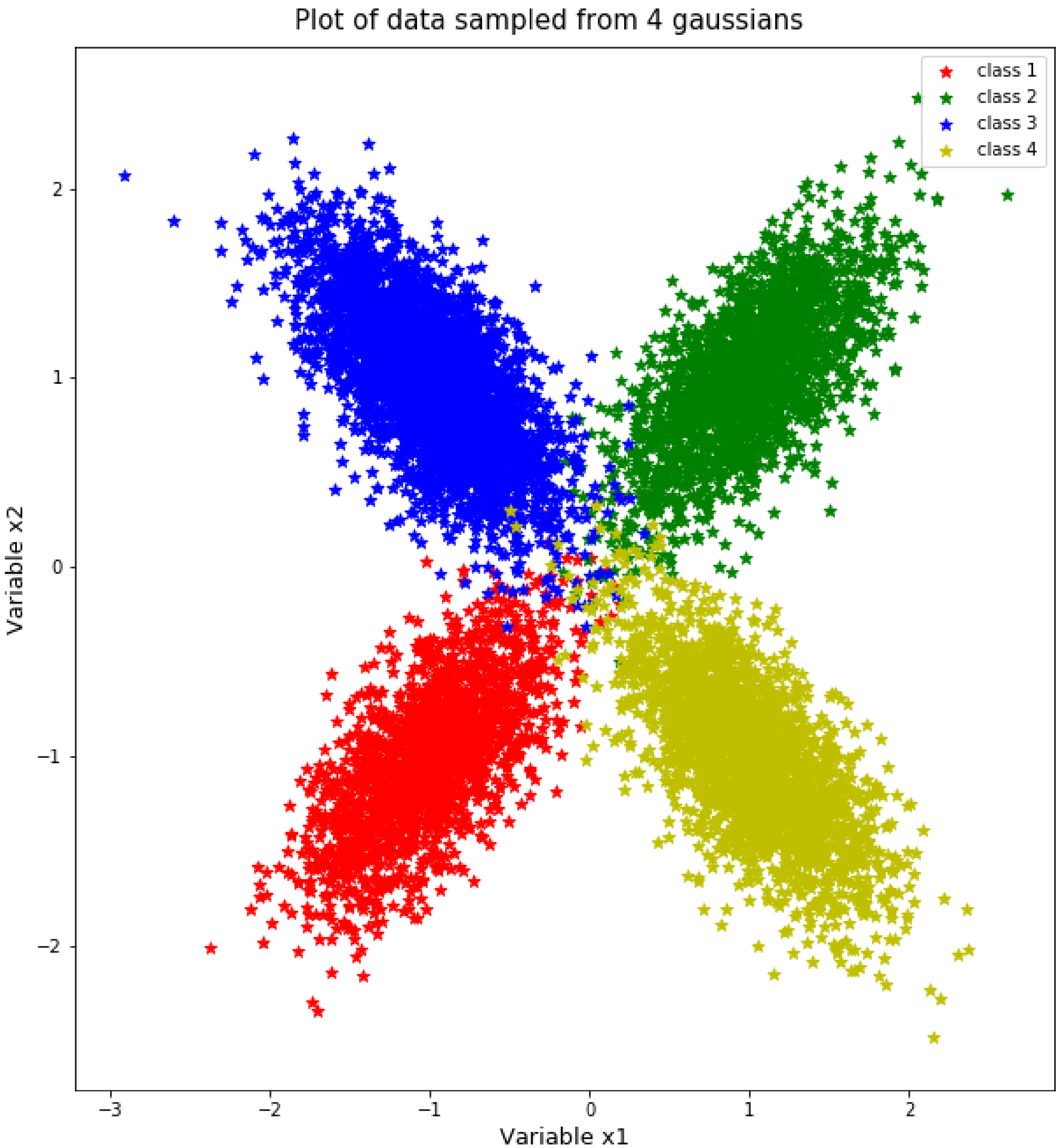

```
Number of samples:  1000
```

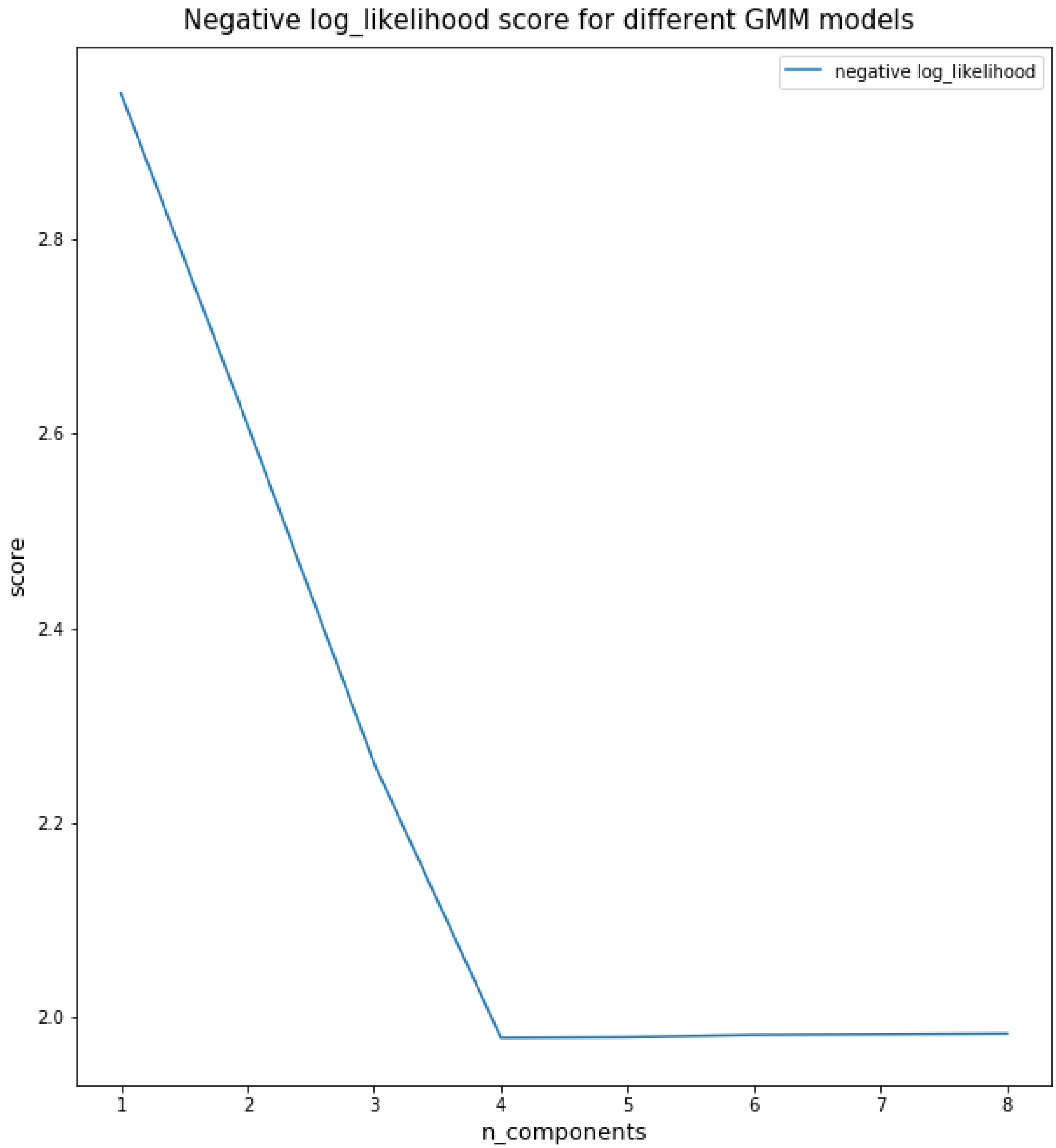

Plot of data sampled from 4 gaussians
class 1
class 2
class 3
class 4
Variable x2
Variable x1
2
1
0
−1
−2
−2
−1
0
1
2

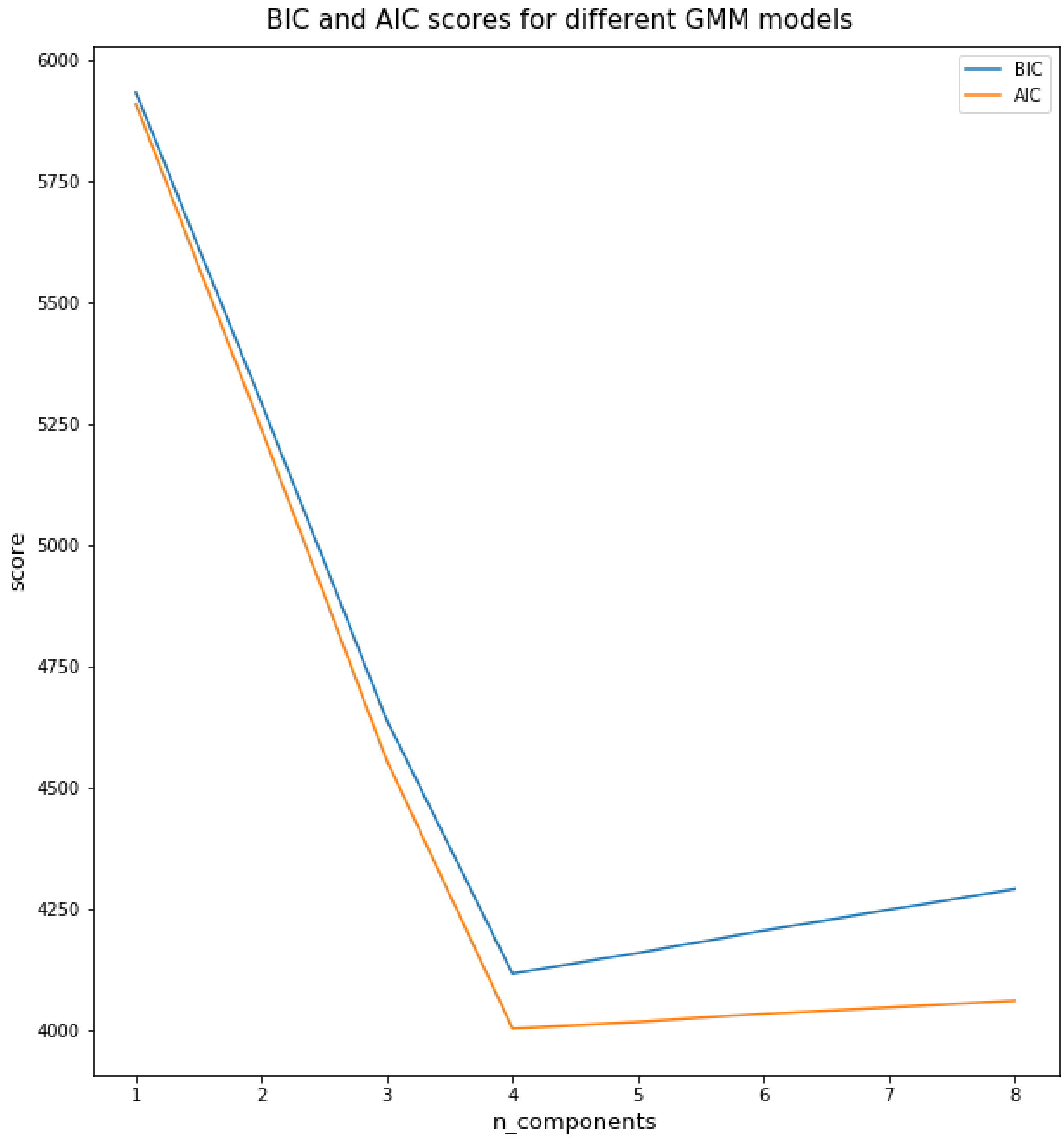

Negative log_likelihood score for different GMM models
negative log_likelihood
score
2.8
2.6
2.4
2.2
2.0
1
2
3
4
5
6
7
8
n_components




```
Number of samples:  10000
```


Plot of data sampled from 4 gaussians
class 1
class 2
class 3
class 4
Variable x2
Variable x1
−3
−2
−1
0
1
2


Negative log_likelihood score for different GMM models
negative log_likelihood
score
2.8
2.6
2.4
2.2
2.0
1
2
3
4
5
6
7
8
n_components



Conclusion

With number of samples=1000 or 10000, log likelihood and bic/aic indicate number of components = 4 is a good model for this data. Further, at n_components=4, the score quicly falls and plateaus around 4. Considering Occam's razor, number of components = 4 is clearly the best option

Answer 2

In [7]:
```
from scipy import linalg
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
from matplotlib import colors

from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

In [8]:
```
cmap = colors.LinearSegmentedColormap(
    'red_blue_classes',
    {'red': [(0, 1, 1), (1, 0.7, 0.7)],
     'green': [(0, 0.7, 0.7), (1, 0.7, 0.7)],
     'blue': [(0, 0.7, 0.7), (1, 1, 1)]})
plt.cm.register_cmap(cmap=cmap)
```

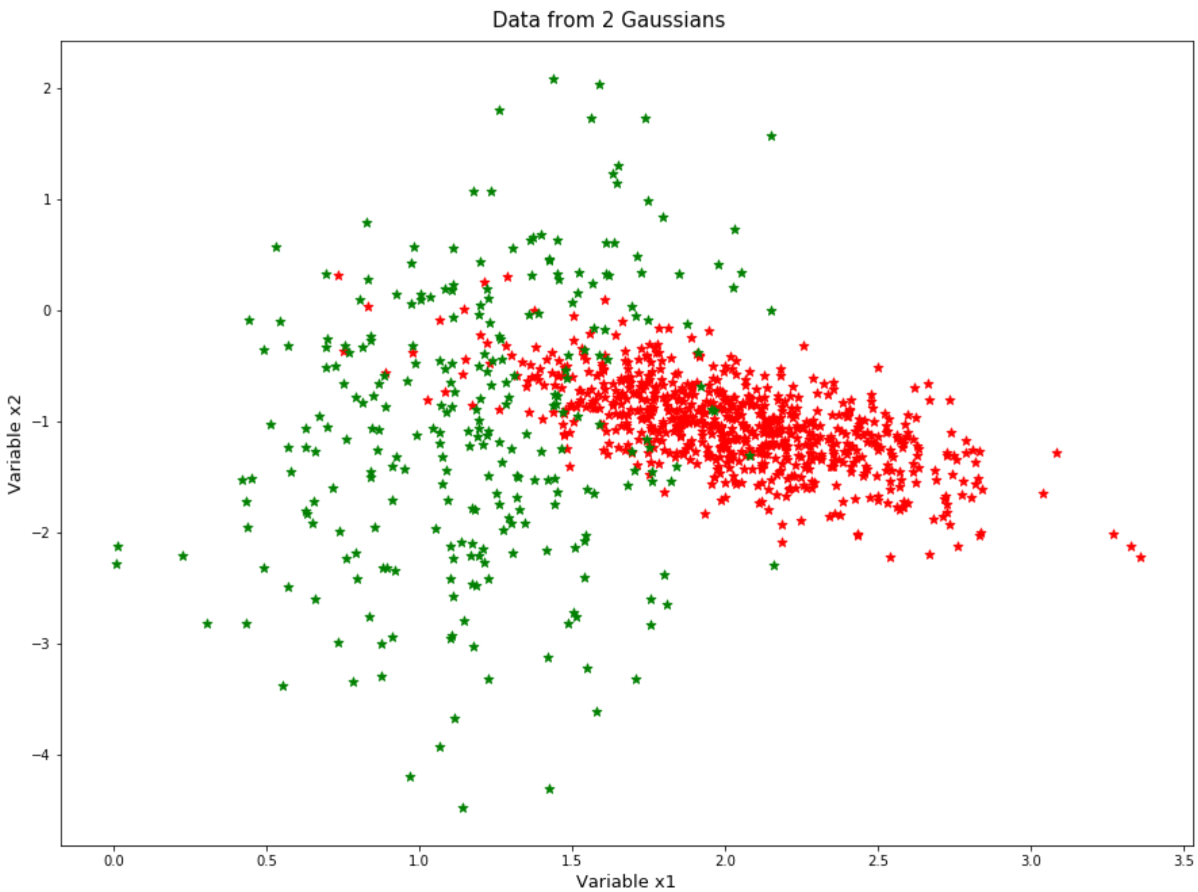
In [9]:
```
def generate_GMM_samples_2class(prior,number_of_samples,sig1,sig2,u1,u2
):
    '''
    Args:
    prior of class 1 = prior[0]
    prior of class 2 = 1-prior[0]


    number_of_samples

    class 1- u_1, sig_1
    class 2- u_2, sig_2


    x is samples from zero-mean identity-covariance Gaussian sample gene
rators

    generating class 1- A1*x+b1
    generating class 2- A2*x+b2


    '''
    from matplotlib.pyplot import figure
    txt="Plot of data sampled from 4 gaussians "
    fig = plt.figure(figsize=(15,10));
    #fig.text(.35,0.09,txt,fontsize=15);

    samples_class1=[]
    samples_class2=[]


    sig_1=np.matrix(sig1)
    sig_2=np.matrix(sig2)


    u_1=np.matrix(u1).transpose()
    u_2=np.matrix(u2).transpose()

    prior=prior
    A1=np.linalg.cholesky(sig_1)
    b1=u_1

    A2=np.linalg.cholesky(sig_2)
    b2=u_2

    zero_mean=[0,0]
    cov=[[1,0],[0,1]]

    for i in range(number_of_samples):
        uniform_sample=np.random.uniform()

        sample_from_zero_mean_identity_covariance=np.random.multivariate
_normal(zero_mean,cov,[1]).transpose()

        if uniform_sample<prior[0]:
            '''sample from class class 1'''
```

```
            sample=A1.dot(sample_from_zero_mean_identity_covariance)+b1
            samples_class1.append(sample)
        elif (prior[0]<uniform_sample<prior[0]+prior[1]):
            '''sample from class class 2'''
            sample=A2.dot(sample_from_zero_mean_identity_covariance)+b2
            samples_class2.append(sample)

    samples_class1_final=np.hstack(samples_class1)
    samples_class2_final=np.hstack(samples_class2)


    a=np.squeeze(np.asarray(samples_class1_final.transpose()[:,1]))
    b=np.squeeze(np.asarray(samples_class1_final.transpose()[:,0]))

    c=np.squeeze(np.asarray(samples_class2_final.transpose()[:,1]))
    d=np.squeeze(np.asarray(samples_class2_final.transpose()[:,0]))
    plt.xlabel('Variable x1',size=13)
    plt.ylabel('Variable x2',size=13)
    fig.suptitle('Data from 2 Gaussians', fontsize=15)
    plt.scatter(b,a,color='r',marker='*',label='class 1',s=50)
    plt.scatter(d,c,color='g',marker='*',label='class 2',s=50)
    X=np.hstack([samples_class1_final,samples_class2_final])
    plt.subplots_adjust(top=.95)
    y = np.hstack((np.zeros(samples_class1_final.shape[1]), np.ones(samp
les_class2_final.shape[1])))
    X=X.T
    X=np.squeeze(np.asarray(X))
    return X,y
```

True GMM

In [10]:
```
# prior of Gaussian 1 = prior[0]
# prior of Gaussian 2 = prior[1]= 1-prior[0]
prior=[0.7,0.3]

# mean and covariance of gaussian 1
sig1=[[.15,.1],[-.1,.15]]
u1=[2,-1]

# mean and covariance of gaussian 2
sig2=[[.15,.1],[.1,1.5]]
u2=[1.2,-1]
X,y=generate_GMM_samples_2class(prior=prior,number_of_samples=999,sig1=s
ig1,sig2=sig2,\
                                u1=u1,u2=u2);

print ("Samples from GMM :",X.shape)
print ("Class labels of the GMM :",y.shape)
```

```
Samples from GMM : (999, 2)
Class labels of the GMM : (999,)
```



LDA

In [11]:
```
def plot_data(lda, X, y, y_pred):
    '''
    LDA helper function
    '''
    plt.xlabel('Variable x1',size=13)
    plt.ylabel('Variable x2',size=13)
    fig.suptitle('LDA decision boundary for data from 2 Gaussians', font
size=15)
    tp = (y == y_pred)  # True Positive
    tp0, tp1 = tp[y == 0], tp[y == 1] # True Positive for class 0 and cl
ass 1 respectively
    # tp0 is boolean with "tp and class=0"
    X0, X1 = X[y == 0], X[y == 1] # points
    X0_tp, X0_fp = X0[tp0], X0[~tp0]
    X1_tp, X1_fp = X1[tp1], X1[~tp1]

    # class 0: dots
    plt.scatter(X0_tp[:, 0], X0_tp[:, 1], marker='.', color='red')
    plt.scatter(X0_fp[:, 0], X0_fp[:, 1], marker='x',
                s=20, color='#990000')  # dark red

    # class 1: dots
    plt.scatter(X1_tp[:, 0], X1_tp[:, 1], marker='.', color='blue')
    plt.scatter(X1_fp[:, 0], X1_fp[:, 1], marker='x',
                s=20, color='#000099')  # dark blue

    # class 0 and 1 : areas
    nx, ny = 200, 100
    x_min, x_max = plt.xlim()
    y_min, y_max = plt.ylim()
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, nx),
                         np.linspace(y_min, y_max, ny))
    Z = lda.predict_proba(np.c_[xx.ravel(), yy.ravel()])
    Z = Z[:, 1].reshape(xx.shape)
    plt.pcolormesh(xx, yy, Z, cmap='red_blue_classes',
                   norm=colors.Normalize(0., 1.), zorder=0)
    plt.contour(xx, yy, Z, [0.5], linewidths=2., colors='white')

    # means
    plt.plot(lda.means_[0][0], lda.means_[0][1],
             '*', color='yellow', markersize=15, markeredgecolor='grey')
    plt.plot(lda.means_[1][0], lda.means_[1][1],
             '*', color='yellow', markersize=15, markeredgecolor='grey')

    return
```

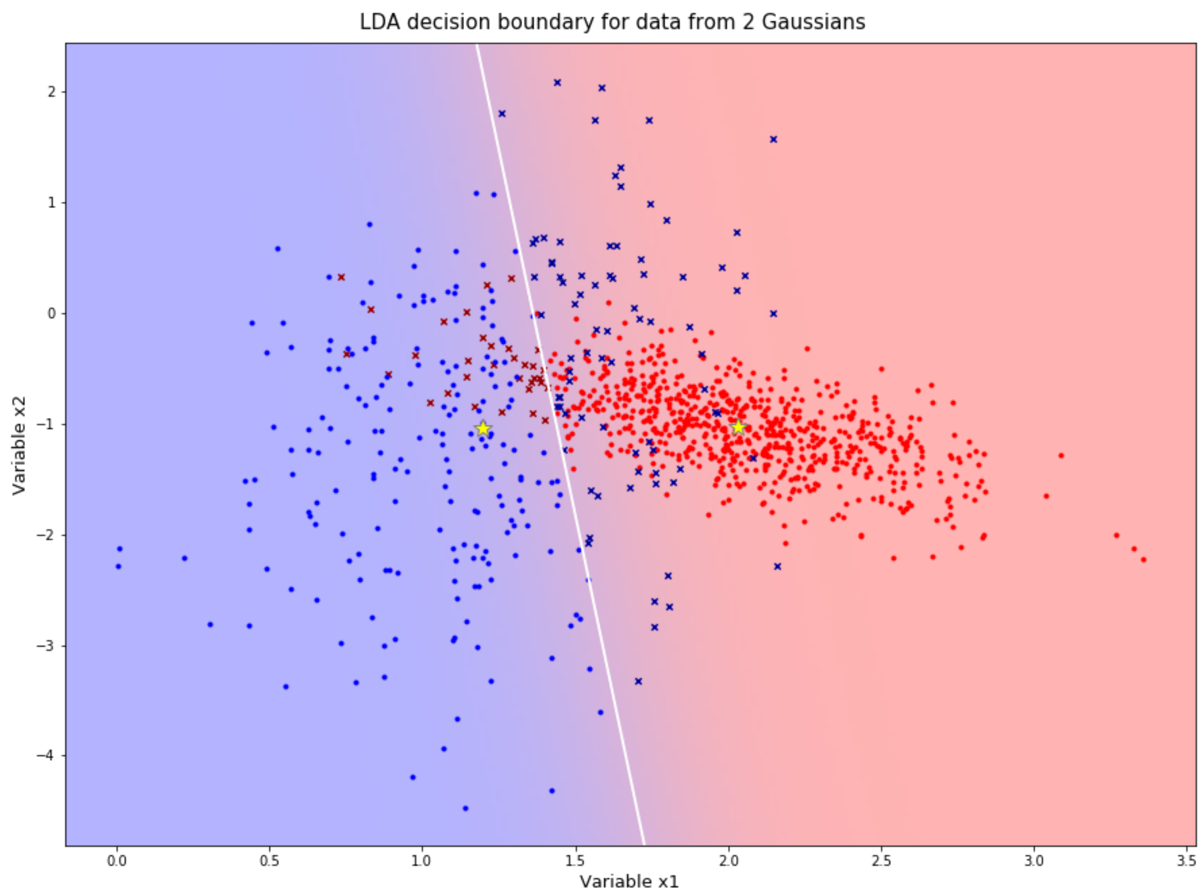
In [12]:
```
lda = LinearDiscriminantAnalysis(store_covariance=True)
lda.fit(X, y)
y_pred = lda.predict(X)
print ("Accuracy with LDA is :",np.mean(y_pred==y))
```

```
Accuracy with LDA is : 0.8858858858858859
```

In [13]:
```
'''
> To Predict-
>> ((np.matmul(X,lda.coef_.T)+ lda.intercept_)>0).astype(int)
>>or
>> lda.predict(X).reshape(-1,1);
''';

# LDA coefficients are-
w=lda.coef_.T
b=lda.intercept_
print (w,b)
```

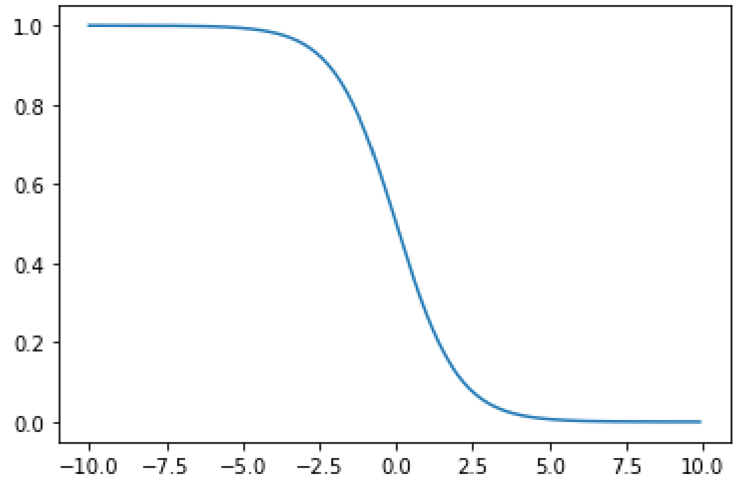
```
[[-5.531773]
 [-0.419741]] [7.53545276]
```

In [14]:
```
from matplotlib.pyplot import figure
fig = plt.figure(figsize=(15,10));
fig.suptitle('Linear Discriminant Analysis', fontsize=15)
plot_data(lda, X, y, y_pred)
plt.subplots_adjust(top=.95)
plt.show()
```



Logistic Regression

In [15]:
```
def test_logistic_function(x):
    return 1/(1+np.exp(x))
x=np.arange(-10,10,.1)
plt.plot(x,test_logistic_function(x));
```



Answer 2

$$\rightarrow W_{LDA} \cdot X + b_{LDA} \underset{\text{class } 0}{\overset{\text{class } 1}{\gtrless}} 0$$

$$y(x) = \frac{1}{1 + e^{(w^T x + b)}}$$

$$\begin{pmatrix} W_{MLE} \\ b_{MLE} \end{pmatrix} = \underset{\begin{pmatrix} w \\ L \end{pmatrix}}{\operatorname{argmax}} \; P\left[D \mid w \right]$$

where $D = \begin{bmatrix} x_1, L_1 \\ x_2, L_2 \\ \vdots \\ x_N, L_N \end{bmatrix}$; $X_i \in \mathbb{R}^2 \rightarrow$ given sample

$L_i \in 0$ or 1

as samples are i.i.d $\rightarrow$

$$\begin{pmatrix} W_{MLE} \\ b_{MLE} \end{pmatrix} = \underset{\begin{pmatrix} w \\ b \end{pmatrix}}{\operatorname{argmax}} \sum_{i=1}^{N} \log p\left(\begin{matrix} x_i \\ L_i \end{matrix} \,\middle|\, \begin{matrix} w \\ b \end{matrix} \right)$$

$$\begin{pmatrix} W_{MLE} \\ b_{MLE} \end{pmatrix} = \underset{\begin{pmatrix} w \\ b \end{pmatrix}}{\operatorname{argmax}} \left[\sum_{i=1}^{N} \log p(L_i \mid x_i; w, b) + \sum_{i=1}^{N} \log p(x_i \mid w, b) \right]$$

$$p(x_i \mid w, b) = p(x_i) \quad (\text{independent of } w, b)$$

$$\Rightarrow \begin{pmatrix} W_{MLE} \\ b_{MLE} \end{pmatrix} = \underset{\begin{pmatrix} w \\ b \end{pmatrix}}{\operatorname{argmax}} \sum_{i=1}^{N} \log p(L_i \mid x_i, w, b)$$

$\cancel{p(L_i \mid x_i, w, b)}$ Assuming class labels 0 and 1 $\rightarrow$

$$\Rightarrow \begin{pmatrix} W_{MLE} \\ b_{MLE} \end{pmatrix} = \underset{\begin{pmatrix} w \\ b \end{pmatrix}}{\operatorname{argmax}} \sum_{i=1}^{N} \log \left[\left(y(x_i)^{L_i} \right) \left(1 - y(x_i) \right)^{(1-L_i)} \right]$$

$$= \underset{\begin{pmatrix} w \\ b \end{pmatrix}}{\operatorname{argmax}} \sum_{i=1}^{N} \left[L_i \log y(x_i) + (1 - L_i) \log (1 - y(x_i)) \right]$$

Let $y(x) = \dfrac{1}{1+e^{x}}$ → sigmoid f^n for us.

$$y'(x) = \frac{-1}{(1+e^{x})^2} \times e^{x}$$

$$y'(x) = y(x)\left[\frac{1}{1+e^{x}} - 1\right]$$

$$= y(x)\left[y(x) - 1\right]$$

$$y'(x) = -y(x)\left(1 - y(x)\right)$$

Further represent $\begin{bmatrix} W \\ b \end{bmatrix}$ by a single parameter θ

and add scalar '1' to the feature 'x';
so that $\left[W^T x + b = \theta^T x \right]$

$$\Rightarrow \theta_{MLE} = \underset{\theta}{\operatorname{argmax}} \underbrace{\sum_{i=1}^{N} \left[L_i \log y(\theta^T x) + (1 - L_i) \log (1 - y(\theta^T x)) \right]}_{L(\theta)}$$

To maximize ; use gradient ascent $\rightarrow$

$$\theta = \theta + \alpha \nabla_\theta l(\theta)$$

for 1 training example $\rightarrow$

$$\frac{\partial l \theta}{\partial \theta_j} = \left[\frac{L_i}{g(\theta^T x)} - \frac{(1-L_i)}{[1-g(\theta^T x)]}\right] \frac{\partial g(\theta^T x)}{\partial \theta_j}$$

where $j = 1, 2, 3$

$$\theta = [w_1, w_2, b]$$

$$\frac{\partial l(\theta)}{\partial \theta_j} = -\left[\frac{L_i}{g(\theta^T x)} - \frac{(1-L_i)}{(1-g(\theta^T x))}\right] g(\theta^T x) \times (1 - g(\theta^T x)) \times \frac{\partial \theta^T x}{\partial \theta_j}$$

$$= -\left[L_i(1-g(\theta^T x)) - (1-L_i)\, g(\theta^T x)\right] x_{ij}$$

$$= -\left[L_i - g(\theta^T x)\right] x_{ij} \quad ; \begin{array}{l} i \rightarrow \text{example} \\ j \rightarrow \text{feature number} \end{array}$$

$$\Rightarrow \theta_j = \theta_j - \alpha\left(y^i - y(\theta^T x^i)\right)x_j^i .$$

In [16]:
```
class LogisticRegression:
    def __init__(self, lr=0.01, num_iter=100000, fit_intercept=True, ver
bose=True):
        self.lr = lr
        self.num_iter = num_iter
        self.fit_intercept = fit_intercept
        self.verbose = verbose

    def __add_intercept(self, X):
        intercept = np.ones((X.shape[0], 1))
        return np.concatenate((intercept, X), axis=1)

    def __sigmoid(self, z):
        return 1 / (1 + np.exp(z))
    def __loss(self, h, y):
        return (y * np.log(h) + (1 - y) * np.log(1 - h)).mean()

    def fit(self, X, y):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        # weights initialization
        self.theta = np.zeros(X.shape[1])

        for i in range(self.num_iter):
            z = np.dot(X, self.theta)
            h = self.__sigmoid(z)
            gradient = np.dot(X.T, ( y -h )) / y.size
            self.theta = self.theta - self.lr * gradient

            z = np.dot(X, self.theta)
            h = self.__sigmoid(z)
            loss = self.__loss(h, y)

            if(self.verbose ==True and i % 10000 == 0):
                print(f'loss: {loss} \t')

    def predict_prob(self, X):
        if self.fit_intercept:
            X = self.__add_intercept(X)

        return self.__sigmoid(np.dot(X, self.theta))

    def predict(self, X):
        return self.predict_prob(X).round()
```

In [17]:
```
def plot_logistic():
    '''
    Helper function
    '''
    from matplotlib.pyplot import figure
    fig = plt.figure(figsize=(15,10));
    plt.xlabel('Variable x1',size=13)
    plt.ylabel('Variable x2',size=13)
    fig.suptitle('Logistic Regression decision boundary for data from 2
 Gaussians', fontsize=15)
    tp = (y == y_pred)  # True Positive
    tp0, tp1 = tp[y == 0], tp[y == 1] # True Positive for class 0 and cl
ass 1 respectively
    # tp0 is boolean with "tp and class=0"
    X0, X1 = X[y == 0], X[y == 1] # points
    X0_tp, X0_fp = X0[tp0], X0[~tp0]
    X1_tp, X1_fp = X1[tp1], X1[~tp1]

    # class 0: dots
    plt.scatter(X0_tp[:, 0], X0_tp[:, 1], marker='.', color='red')
    plt.scatter(X0_fp[:, 0], X0_fp[:, 1], marker='x',
                s=20, color='#990000')  # dark red

    # class 1: dots
    plt.scatter(X1_tp[:, 0], X1_tp[:, 1], marker='.', color='blue')
    plt.scatter(X1_fp[:, 0], X1_fp[:, 1], marker='x',
                s=20, color='#000099')  # dark blue

    # class 0 and 1 : areas
    nx, ny = 200, 100
    x_min, x_max = plt.xlim()
    y_min, y_max = plt.ylim()
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, nx),
                         np.linspace(y_min, y_max, ny))
    Z=model.predict_prob((np.c_[xx.ravel(), yy.ravel()]))
    Z=Z.reshape(xx.shape)

    plt.pcolormesh(xx, yy, Z, cmap='red_blue_classes',
                   norm=colors.Normalize(0., 1.), zorder=0)
    plt.contour(xx, yy, Z, [0.5], linewidths=2., colors='white');
    plt.subplots_adjust(top=.95);
```

In [18]:
```
model = LogisticRegression(lr=0.0042, num_iter=int(42000*4.2))
X=np.squeeze(np.asarray(X))
```

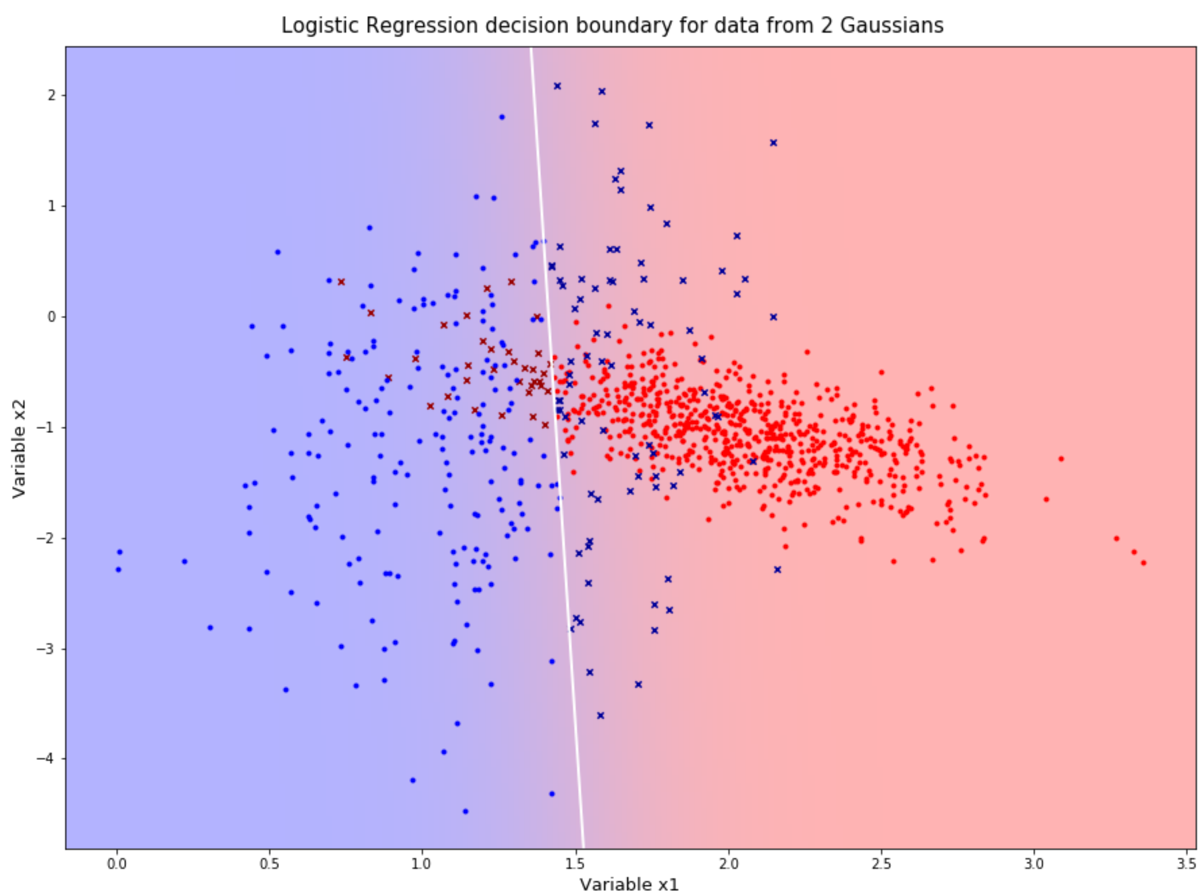
In [19]:
```
model.fit(X, y)
```

```
loss: -0.6913611622242243
loss: -0.3925256110835711
loss: -0.34937699814411055
loss: -0.3271342605257729
loss: -0.31430279830677393
loss: -0.30628444660081783
loss: -0.30098068931434596
loss: -0.29732343894435354
loss: -0.29472103651588316
loss: -0.29282351110119864
loss: -0.2914128817526874
loss: -0.2903476447177061
loss: -0.2895327958119807
loss: -0.2889027510410482
loss: -0.2884111704433018
loss: -0.2880246612358051
loss: -0.28771875344985876
loss: -0.2874752552538597
```

In [20]:
```
y_pred = model.predict(X)
print ("Accuracy of logistic regression is :",(y_pred == y).mean())
```

```
Accuracy of logistic regression is : 0.8828828828828829
```

In [21]:
```
plot_logistic()
```



MAP

Maximize posterior with true priors and probabilties. Select the class with max posterior

In [22]:
```
'''
Helper function
'''

def plot_map(X,y,y_pred):
    from matplotlib.pyplot import figure
    fig = plt.figure(figsize=(15,10));
    plt.xlabel('Variable x1',size=13)
    plt.ylabel('Variable x2',size=13)
    fig.suptitle('MAP decision boundary for data from 2 Gaussians', font
size=15)
    tp = (y == y_pred)  # True Positive
    tp0, tp1 = tp[y == 0], tp[y == 1] # True Positive for class 0 and cl
ass 1 respectively
    # tp0 is boolean with "tp and class=0"
    X0, X1 = X[y == 0], X[y == 1] # points
    X0_tp, X0_fp = X0[tp0], X0[~tp0]
    X1_tp, X1_fp = X1[tp1], X1[~tp1]

    # class 0: dots
    plt.scatter(X0_tp[:, 0], X0_tp[:, 1], marker='.', color='red')
    plt.scatter(X0_fp[:, 0], X0_fp[:, 1], marker='x',
                s=20, color='#990000')  # dark red

    # class 1: dots
    plt.scatter(X1_tp[:, 0], X1_tp[:, 1], marker='.', color='blue')
    plt.scatter(X1_fp[:, 0], X1_fp[:, 1], marker='x',
                s=20, color='#000099')  # dark blue

    # class 0 and 1 : areas
    nx, ny = 200, 100
    x_min, x_max = plt.xlim()
    y_min, y_max = plt.ylim()
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, nx),
                         np.linspace(y_min, y_max, ny))


    Z = (gaussian1.pdf((np.c_[xx.ravel(), yy.ravel()]))*prior[1])/((gaus
sian2.pdf((np.c_[xx.ravel(), yy.ravel()]))*prior[1])+(gaussian1.pdf((np.
c_[xx.ravel(), yy.ravel()]))*prior[1]))
    Z=Z.reshape(xx.shape)

    plt.pcolormesh(xx, yy, Z, cmap='red_blue_classes',
                   norm=colors.Normalize(0., 1.), zorder=0)
    plt.contour(xx, yy, Z, [0.5], linewidths=2., colors='white');
    plt.subplots_adjust(top=.95);
```

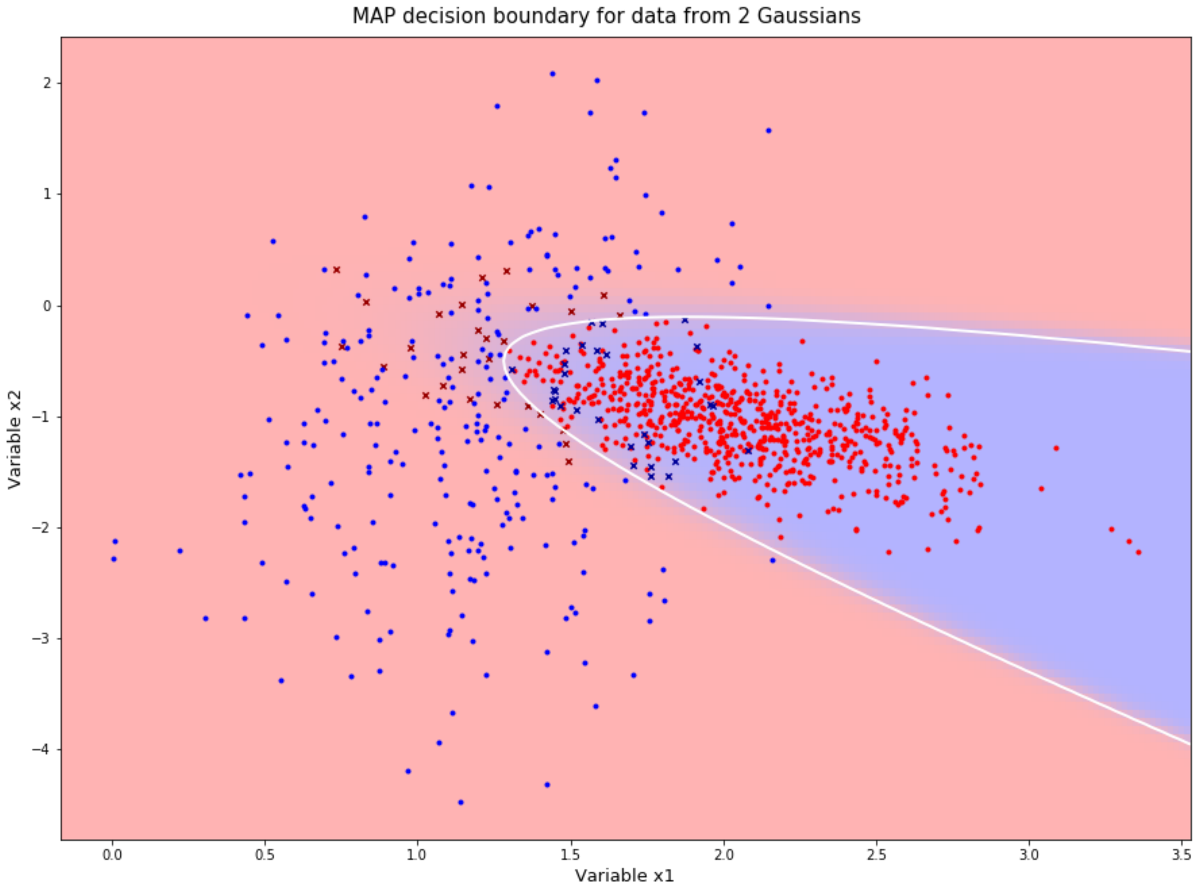
In [23]:
```
# Re writing true priors-

# prior of Gaussian 1 = prior[0]
# prior of Gaussian 2 = prior[1]= 1-prior[0]
prior=[0.7,0.3]

# mean and covariance of gaussian 1
sig1=[[.15,.1],[-.1,.15]]
u1=[2,-1]

# mean and covariance of gaussian 2
sig2=[[.15,.1],[.1,1.5]]
u2=[1,-1]


print ("Samples from GMM :",X.shape)
print ("Class labels of the GMM :",y.shape)
```

```
Samples from GMM : (999, 2)
Class labels of the GMM : (999,)
```

In [24]:
```
gaussian1 = multivariate_normal(mean=u1, cov=sig1)
gaussian2 = multivariate_normal(mean=u2, cov=sig2)

class1_posterior=(gaussian1.pdf(X)*prior[0])/((gaussian1.pdf(X)*prior[0
])+(gaussian2.pdf(X)*prior[0]))
class2_posterior=(gaussian2.pdf(X)*prior[0])/((gaussian1.pdf(X)*prior[0
])+(gaussian2.pdf(X)*prior[0]))

class1_class2_posterior_stacked=np.vstack([class1_posterior,class2_poste
rior]).T

y_pred=np.argmax(class1_class2_posterior_stacked,axis=1)
print ("Accuracy of MAP is :",np.mean(y==y_pred))
```

```
Accuracy of MAP is : 0.9419419419419419
```

In [25]:
```
plot_map(X,y,y_pred)
```



Conclusion

MAP classifier has the highest accuracy of close to 94%. LDA and logistic regression have accuracy close to 88%. The decision boundaries of 3 classifiers are shown above

References

1. Lecture notes (https://github.com/rohinarora/EECE5644-Machine_Learning)
2. https://scikit-learn.org/0.16/auto_examples/classification/plot_lda_qda.html (https://scikit-learn.org/0.16/auto_examples/classification/plot_lda_qda.html)
3. https://github.com/martinpella/logistic-reg/blob/master/logistic_reg.ipynb (https://github.com/martinpella/logistic-reg/blob/master/logistic_reg.ipynb)
4. http://cs229.stanford.edu/notes/cs229-notes1.pdf (http://cs229.stanford.edu/notes/cs229-notes1.pdf)
5. MinM (https://en.wikipedia.org/wiki/Eminem)
6. Don't Panic (https://en.wikipedia.org/wiki/Don't_Panic:_The_Official_Hitchhiker's_Guide_to_the_Galaxy_Companion)