In [1]:
```
# Use gammaln for stability
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from scipy.io import loadmat
from scipy.special import digamma, gammaln, multigammaln
from scipy.stats import multivariate_normal, wishart
from sklearn.covariance import empirical_covariance
```

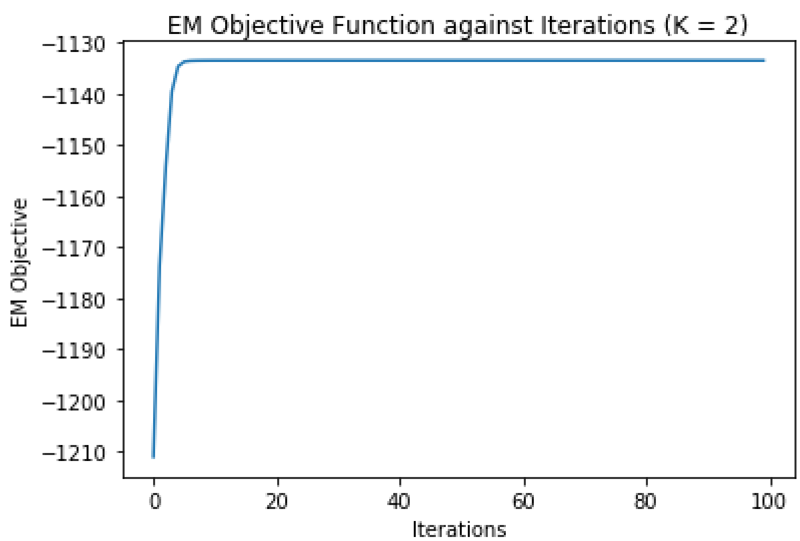
In [2]:
```
# Load data
data = loadmat('hw4_data_mat/data.mat')
X = data['X']
d = X.shape[0]
num = X.shape[1]
```

In [3]:
```
def EM_GMM(X, k):
    # Initialise
    pi = np.ones(k)
    mu = np.random.rand(d, k)
    lamda = [np.identity(d) for i in range(k)]
    LL = []

    for a in range(100):
        # E-Step
        c = np.empty((k, num))
        for i in range(k):
            c[i, :] = map(lambda j: pi[i] * multivariate_normal.pdf(X[:,
 j], mu[:, i], np.linalg.inv(lamda[i])), range(num))
        for j in range(num):
            c[:, j] = c[:, j] / float(np.sum(c[:, j]))

        # M-Step
        n = np.sum(c, axis=1)
        for i in range(k):
            mu[:, i] = 1/float(n[i]) * np.dot(X, c[i, :].T)
            x_minus_mu_j = X.T - mu[:, i]
            Sigma = 1/float(n[i]) * sum(map(lambda j: c[i, j] *
(np.dot(x_minus_mu_j[j].reshape((d, 1)), x_minus_mu_j[j].reshape((1,
d)))), range(num)))
            lamda[i] = np.linalg.inv(Sigma)
        pi = n / float(250)

        # Calculate log-likelihood
        LL_t = 0
        for i in range(num):
            LL_t += np.log(sum(map(lambda j: pi[j] * multivariate_normal.
(X[:, i], mu[:, j], np.linalg.inv(lamda[j])), range(k))))
        LL.append(LL_t)

    return LL, c
```

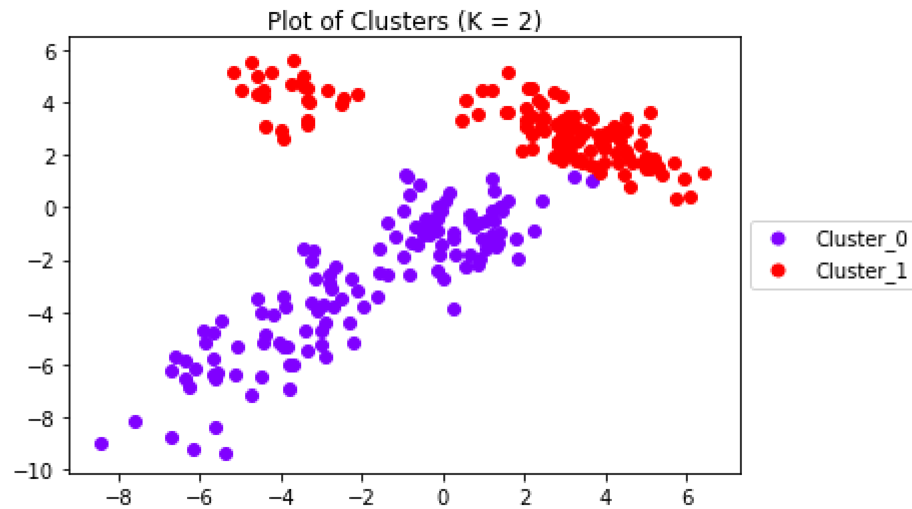
In [4]:
```
def plot_clusters(X, c, k):
    cluster = {}
    for i in range(k):
        cluster[i] = [[], []]
    for i in range(250):
        assignment = np.argmax(c[:, i])
        cluster[assignment][0].append(X[:, i][0])
        cluster[assignment][1].append(X[:, i][1])
    color = iter(plt.cm.rainbow(np.linspace(0,1,k)))
    for i in range(k):
        plt.scatter(cluster[i][0], cluster[i][1], label='Cluster_' + str(
), c=next(color), marker='o')
    plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
    plt.title('Plot of Clusters (K = ' + str(k) + ')')
```

In [5]:
```
L_2, c_2 = EM_GMM(X, 2)
plt.plot(range(100), L_2)
plt.xlabel('Iterations')
plt.ylabel('EM Objective')
plt.title('EM Objective Function against Iterations (K = 2)')
```

Out[5]: Text(0.5,1,u'EM Objective Function against Iterations (K = 2)')

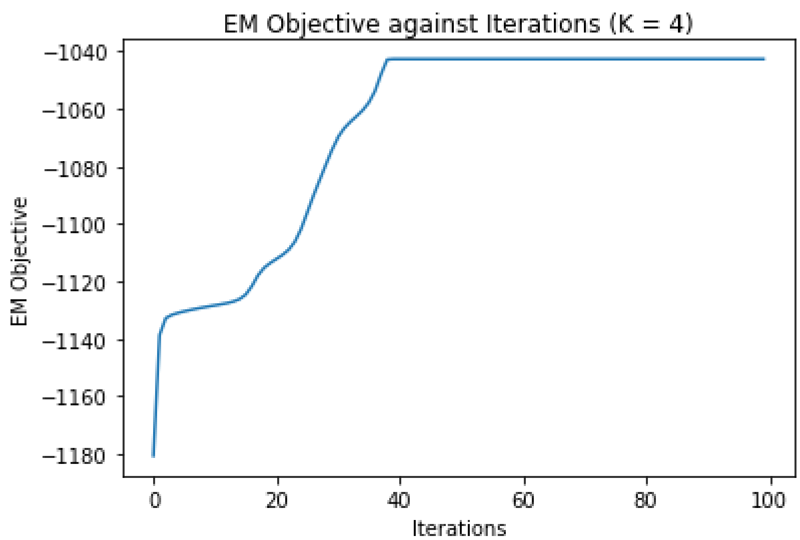


In [6]:
```
plot_clusters(X, c_2, 2)
```

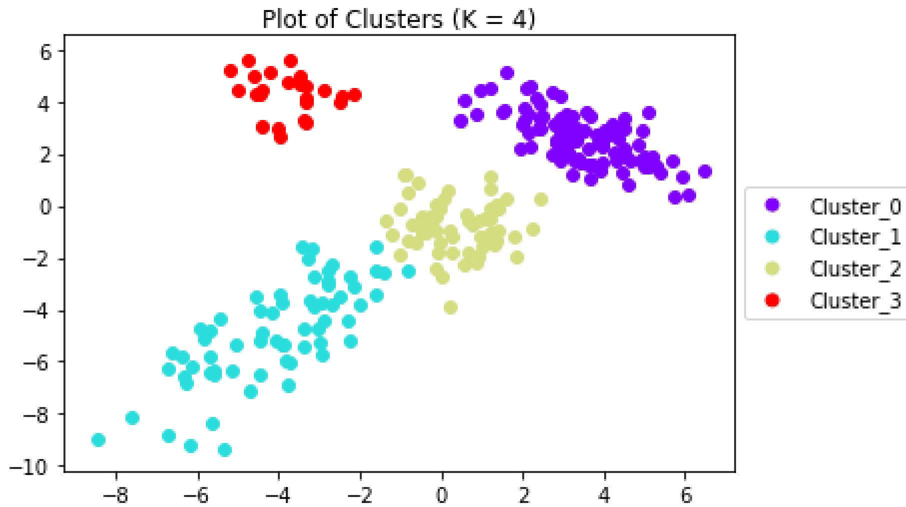


In [7]:
```
L_4, c_4 = EM_GMM(X, 4)
plt.plot(range(100), L_4)
plt.xlabel('Iterations')
plt.ylabel('EM Objective')
plt.title('EM Objective against Iterations (K = 4)')
```

Out[7]:
```
Text(0.5,1,u'EM Objective against Iterations (K = 4)')
```

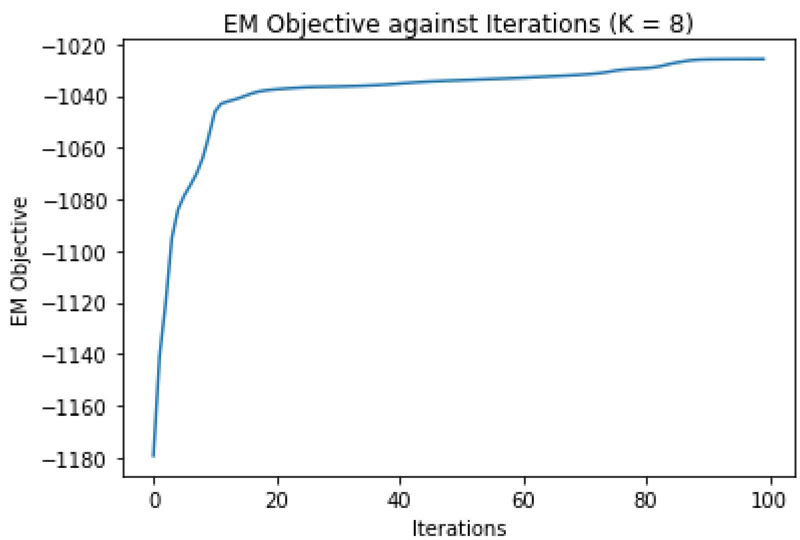


In [8]:
```
plot_clusters(X, c_4, 4)
```

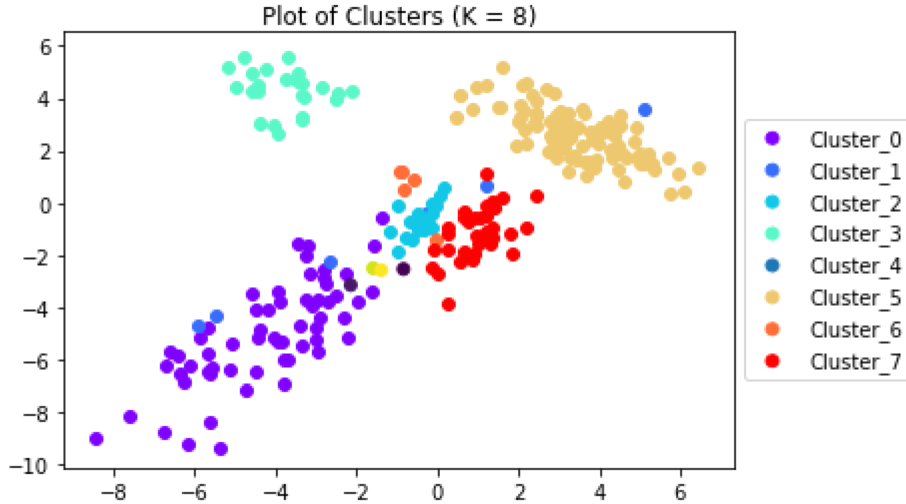


In [9]:
```
L_8, c_8 = EM_GMM(X, 8)
plt.plot(range(100), L_8)
plt.xlabel('Iterations')
plt.ylabel('EM Objective')
plt.title('EM Objective against Iterations (K = 8)')
```

Out[9]:
```
Text(0.5,1,u'EM Objective against Iterations (K = 8)')
```

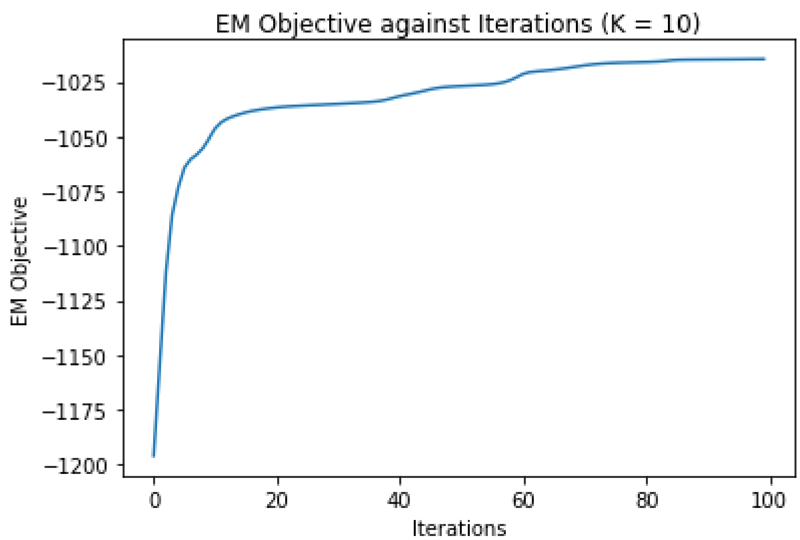


In [10]:
```
plot_clusters(X, c_8, 8)
```

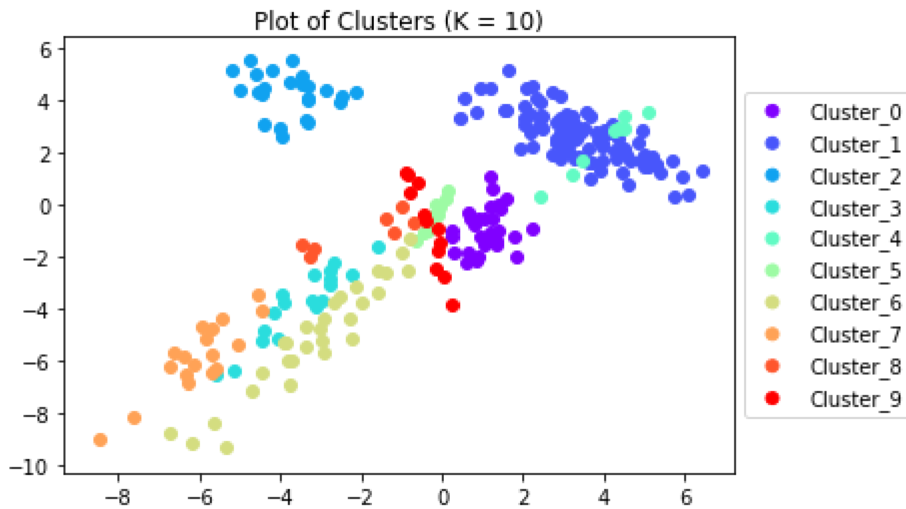


In [11]:
```
L_10, c_10 = EM_GMM(X, 10)
plt.plot(range(100), L_10)
plt.xlabel('Iterations')
plt.ylabel('EM Objective')
plt.title('EM Objective against Iterations (K = 10)')
```

Out[11]:
```
Text(0.5,1,u'EM Objective against Iterations (K = 10)')
```



```
In [12]: plot_clusters(X, c_10, 10)
```




```
In [ ]:
```