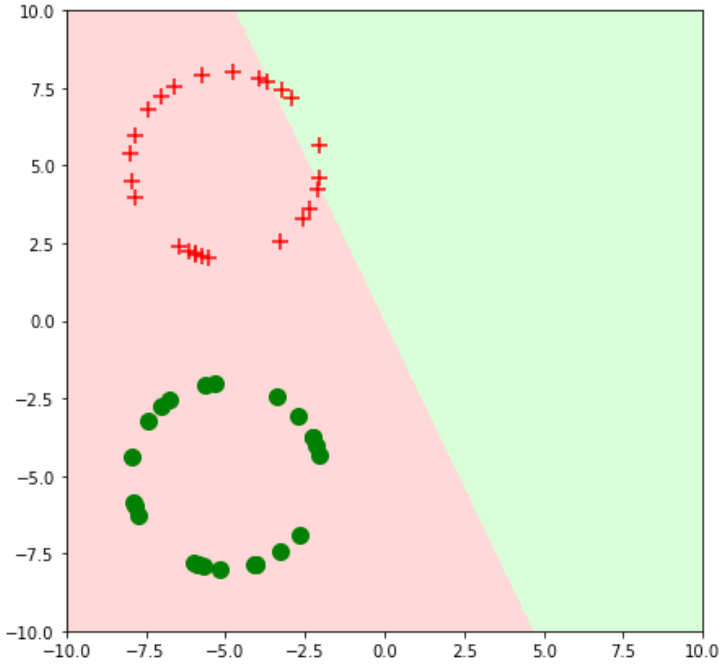
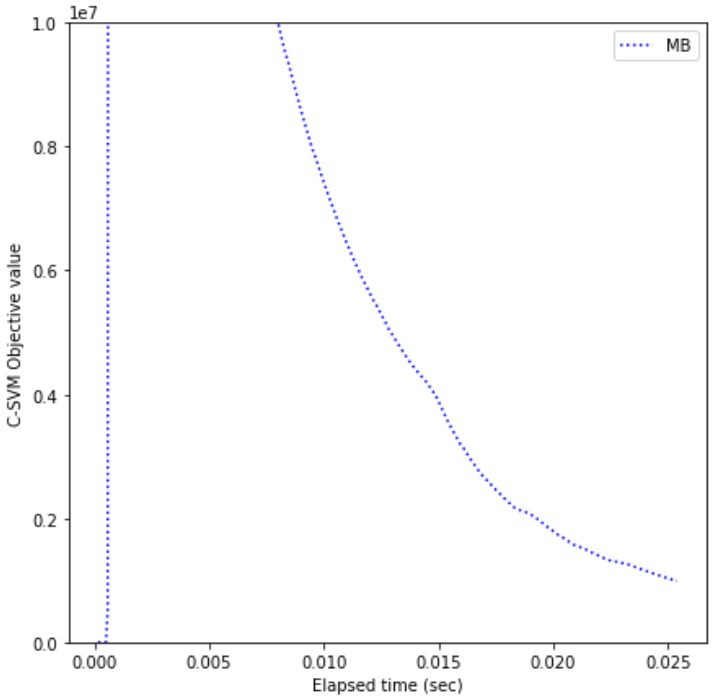
In [216]:
```
import numpy as np
from cs771 import genSyntheticData as gsd
from cs771 import plotData as pd
from matplotlib import pyplot as plt
import time as tm
import random
# Dataset 2 - set of two circles with a couple of outliers
# Comment this section out in order to try dataset 1

muPos1 = np.array( [-5,5] )
muPos2 = np.array( [1,-5] )
muNeg = np.array( [-5,-5] )
r = 3

# Set n to be a large number to visualize the speed benefits of SGD/MB over
GD
d = 2
n = 25


tmp1 = gsd.genSphericalData( d, n, muPos1, r)
tmp2 = gsd.genSphericalData( d, n//30, muPos2, r//2 )
XPos = np.vstack( (tmp1, tmp2) )
XNeg = gsd.genSphericalData( d, n, muNeg, r )
yPos = np.ones( (n + n//30,) )
yNeg = -np.ones( (n,) )
X = np.vstack( (XPos, XNeg) )
y = np.concatenate( (yPos, yNeg) )

np.seterr(all='warn')

# Get a mini-batch stochastic gradient for CSVM
# Choose a random set of B samples per iteration
def getCSVMMBGrad( theta ):
    w = theta#[0:-1]
    #b = theta[-1]
    n = y.size
    if B <= n:
        samples = random.sample( range(0, n), B )
        X_ = X[samples,:]   #10X2
        y_ = y[samples]    #1X 10
    else:
        X_ = X
        y_ = y
    d1 = np.finfo(dtype=np.float64)
    d2 = np.finfo(dtype=np.float64)
    d1 = np.dot( X_.dot( w ) ,( y_ )  )

    d2 =  (-1) * (X_.T ).dot( y_)
    #print(d1)
    #discriminant = np.multiply( (X_.dot( w )+ b ), y_ )
    #g = np.zeros( (B,) )
    #g[discriminant < 1] = -1
    #delb = C * g.dot( y_ )
    #print(np.dot( X_.dot( w ) , y_  ))

    # X_ 10X2 y_ 10X w 2X

    delw = np.finfo(dtype=np.float64)
    delw = w + 2 * C * n/B  - 2* C * ( (n/B) * ( d1 * d2 ) )
    #print(delw)
    return delw #np.append( delw , delb )

# Quite standard for strongly convex but non-smooth objectives like CSVM
def getStepLength( grad, t ):
    return eta/(t+1)
```


10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


1e7
1.0
0.8
0.6
0.4
0.2
0.0
MB
C-SVM Objective value
0.000
0.005
0.010
0.015
0.020
0.025
Elapsed time (sec)

In [ ]:

In [ ]:

In [ ]: