In [5]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
```

Problem 1

Preamble

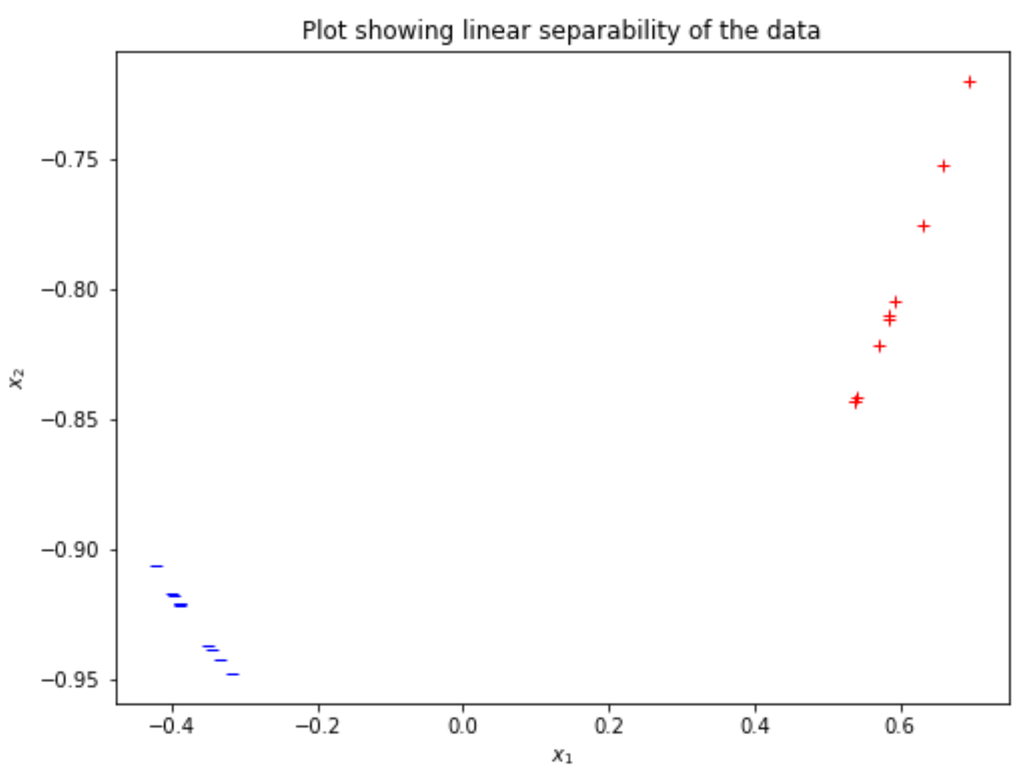
In [6]:
```
# Replace with file path of Problem1.csv if not in the same folder as th
is notebook
prob1_fp = 'Problem1.csv'

# Import data
prob1_df = pd.read_csv(prob1_fp, header=None, names=['x1','x2','y'])

# Set up problem parameters and vectors
n, p, x1, x2, y = prob1_df.shape[0], prob1_df.shape[1] - 1, prob1_df['x
1'], prob1_df['x2'], prob1_df['y']
X = prob1_df[['x1','x2']]


# Preview the first 5 rows
prob1_df.head(5)
```

Out[6]:

	x1	x2	y
0	0.65865	-0.75245	1
1	0.53805	-0.84291	1
2	-0.34941	-0.93697	-1
3	0.56989	-0.82172	1
4	-0.42218	-0.90651	-1

Problem 1 Q1 - Plotting

In [7]:
```
fig_prob1q1, ax = plt.subplots(1, figsize=(8,6))
y_marker = {1: 'r+', -1: 'b_'}
ax.set_title(r'Plot showing linear separability of the data')
ax.set_xlabel(r'$x_1$')
ax.set_ylabel(r'$x_2$')

########################################################################
######
# TODO: Plot the labeled data by looking up the y_marker dictionary, ref
er   #
# to: https://matplotlib.org/devdocs/api/_as_gen/matplotlib.pyplot.plot.
html #
########################################################################
######

for i in range(n):
    if(y[i]<0):
        ax.plot(x1[i],x2[i],'b_', label="Negative")
    else:
        ax.plot(x1[i],x2[i],'r+', label="Positive")

########################################################################
######
#                              END OF YOUR CODE
     #
########################################################################
######

plt.savefig("problem1_q1", dpi=300, bbox_inches='tight') # Exports the f
igure
plt.show()
```



Problem 1 Q2 - Finding max-margin solution using SVM

- We recommend using the *quadprog* module in conjuction with the below function (which is just a convenient wrapper) to solve the SVM optimization. Before running the below cell, install Python package *quadprog* by opening anaconda command prompt and running:

  pip install quadprog

- You can use other optimization packages such as CVXOPT or CVXPY if you are able to install them correctly (if you are using later versions of Python > 3.4 you may need to create virtual environment to install such packages)
- The below function solves a quadratic program:

```
minimize
    (1/2) * x.T * P * x + q.T * x
subject to
    G * x <= h
    A * x == b
```

- **Important:** You might encounter an error (even in other convex optimization software) that the objective matrix is not PD. A workaround is to perturb the matrix along the diagonal by a small positive epsilon.

In [8]:
```
import quadprog
import matrix
def quadprog_solve_qp(P, q, G=None, h=None, A=None, b=None, initvals=Non
e):
    """
    Solve a Quadratic Program defined as:
        minimize
            (1/2) * x.T * P * x + q.T * x
        subject to
            G * x <= h
            A * x == b
    using quadprog <https://pypi.python.org/pypi/quadprog/>.
    Parameters
    ----------
    P : numpy.array
        Symmetric quadratic-cost matrix.
    q : numpy.array
        Quadratic-cost vector.
    G : numpy.array
        Linear inequality constraint matrix.
    h : numpy.array
        Linear inequality constraint vector.
    A : numpy.array, optional
        Linear equality constraint matrix.
    b : numpy.array, optional
        Linear equality constraint vector.
    initvals : numpy.array, optional
        Warm-start guess vector (not used).
    Returns
    -------
    x : numpy.array
        Solution to the QP, if found, otherwise ``None``.
    Note
    ----
    The quadprog solver only considers the lower entries of `P`, therefo
re it
    will use a wrong cost function if a non-symmetric matrix is provide
d.
    """
    if initvals is not None:
        print("quadprog: note that warm-start values ignored by wrapper"
)
    qp_G = P
    qp_a = -q
    if A is not None:
        qp_C = -np.vstack([A, G]).T
        qp_b = -np.hstack([b, h])
        meq = A.shape[0]
    else:  # no equality constraint
        qp_C = -G.T
        qp_b = -h
        meq = 0
    return quadprog.solve_qp(qp_G, qp_a, qp_C, qp_b, meq)[0]

########################################################################
######
```

```
#Function to get minimum gamma for gamma_star = min_gamma/||theta_star||
########################################################################
######
def get_min_gamma(n,y,x1,x2,theta):
    min = 10000
    for i in range(n):
        if(y[i]*(theta[0]*x1[i]+theta[1]*x2[i])<min):
            min = y[i]*(theta[0]*x1[i]+theta[1]*x2[i])
    return min
```

In [18]:
```
################################################################################
######
# TODO: Fill in the matrices as per the hard SVM formulation
     #
# Refer to the function above
     #
################################################################################
######
epsilon = 0.00000000001
P = np.eye(p,p)
q = np.zeros(p)
G = np.zeros((n,p))
h = -1*np.ones(n)
for i in range(n):
    for j in range(p):
        if(j==0):
            G[i,j] = (-1*y[i]*x1[i])
        else:
            G[i,j] = (-1*y[i]*x2[i])
################################################################################
######
#                              END OF YOUR CODE
     #
################################################################################
######

theta_star = quadprog_solve_qp(P, q, G=G, h=h)
# Normalize theta
theta_star = (1.0/np.linalg.norm(theta_star)) * theta_star

################################################################################
######
# TODO: Retrieve gamma_star from theta_star
     #
################################################################################
######

gamma_min = get_min_gamma(n,y,x1,x2,theta_star) #get the minimum gamma a
s defined in the previous function

gamma_star = gamma_min/np.linalg.norm(theta_star)

################################################################################
######
#                              END OF YOUR CODE
     #
################################################################################
######

print('Problem 1 Q2 \n=================================')
print('theta^*: %s' % theta_star)
print('gamma^*: %s' % gamma_star)
```

In [19]:
```python
## Std. Perceptron Implementation
"""
Parameters:
X: (n,p) feature matrix
y: n-vector of labels
theta_zero: initial theta

Returns:
k: # updates
theta_k: Converged (normalized) theta
gamma: margin for this theta_k
"""
def std_perceptron(X, y, theta_zero):
    k = 0
    theta_k = theta_zero
    gamma = None
    ########################################################################
##########
    # TODO: Implement the std. perceptron algorithm, update theta_k and
 increment#
    # k when necessary, remember to normalize theta and calculate gamma
     #
    ########################################################################
##########
    i = 0
    x1 = X['x1']
    x2 = X['x2']
    max_iterations = 1000
    updates = 0
    while(updates<max_iterations):
        i = 0
        while(i<len(y)):
            if(y[i]*(theta_k[0]*x1[i]+theta_k[1]*x2[i])<=0):
                k+=1
                theta_k[0] = theta_k[0] + y[i]*x1[i]
                theta_k[1] = theta_k[1] + y[i]*x2[i]
            i+=1
        updates+=1
        if(k>max_iterations):
            print("Max iterations exceeded; data is non-separable!")
            break;

    theta_k = (1.0/np.linalg.norm(theta_k)) * theta_k
    gamma = get_min_gamma(len(y),y,x1,x2,theta_k)/np.linalg.norm(theta_k
)
    ########################################################################
##########
    #                              END OF YOUR CODE
     #
    ########################################################################
##########
    return k, theta_k, gamma
```

In [20]:
```
## Run this to see results
theta_zero = np.zeros(p)
k, theta, gamma = std_perceptron(X, y, theta_zero)
print('Problem 1 Q3(a) \n=================================')
print("Number of updates, k = %s" % k)
print("theta = %s" % theta)
print("gamma = %s" % gamma)
```

```
Problem 1 Q3(a)
=================================
Number of updates, k = 2
theta = [ 0.98365688  0.18005314]
gamma = 0.377454883747
```

Problem 1 Q3(b)

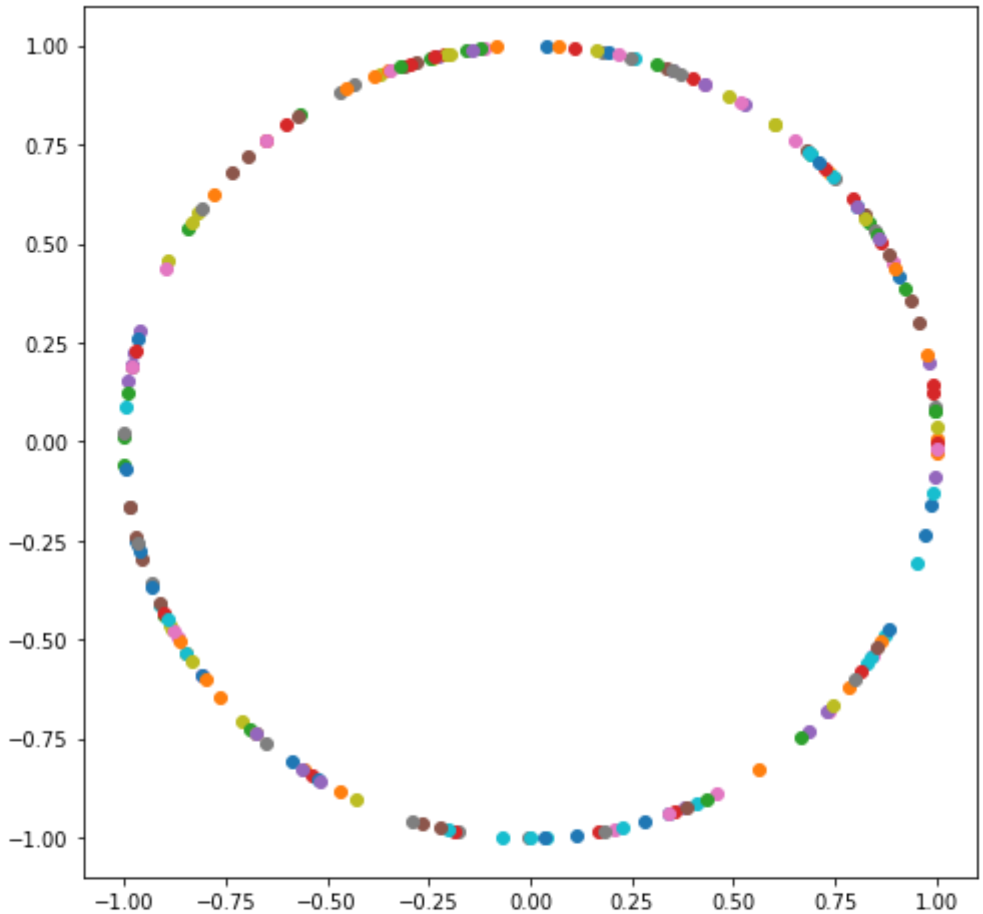
In [21]:
```
num_iter = 10
for i in range(1,num_iter + 1):
    ######################################################################
##########
    # TODO: Generate random theta_zero and run perceptron
         #
    ######################################################################
##########
    # Store the results for this iteration in these variables

    theta_zero_i = np.zeros(p)+(np.random.rand(),np.random.rand())

    print('=================Iter %s==================' % i)
    print('theta_zero = %s' % theta_zero_i)

    k_i,theta_i,gamma_i = std_perceptron(X, y, theta_zero_i)

    print('k = %s, theta = [%.3f,%.3f], gamma=%.3f' % (k_i, theta_i[0],
theta_i[1], gamma_i))

    ######################################################################
##########
    #                               END OF YOUR CODE
         #
    ######################################################################
##########
```

```
==================Iter 1==================
theta_zero = [ 0.74794344  0.79374993]
k = 1, theta = [1.000,0.029], gamma=0.346
==================Iter 2==================
theta_zero = [ 0.12825667  0.29218249]
k = 2, theta = [0.922,0.387], gamma=0.170
==================Iter 3==================
theta_zero = [ 0.77883043  0.0649501 ]
k = 0, theta = [0.997,0.083], gamma=0.396
==================Iter 4==================
theta_zero = [ 0.18658537  0.55412612]
k = 1, theta = [0.974,-0.228], gamma=0.094
==================Iter 5==================
theta_zero = [ 0.65364266  0.00123201]
k = 0, theta = [1.000,0.002], gamma=0.320
==================Iter 6==================
theta_zero = [ 0.07679343  0.86728292]
k = 1, theta = [0.988,0.154], gamma=0.402
==================Iter 7==================
theta_zero = [ 0.73933756  0.65419106]
k = 1, theta = [0.998,-0.070], gamma=0.251
==================Iter 8==================
theta_zero = [ 0.76393958  0.16413424]
k = 0, theta = [0.978,0.210], gamma=0.349
==================Iter 9==================
theta_zero = [ 0.22481565  0.60812585]
k = 1, theta = [0.987,-0.161], gamma=0.162
==================Iter 10==================
theta_zero = [ 0.75852004  0.05519825]
k = 0, theta = [0.997,0.073], gamma=0.386
```

Problem 1 Q3(d)

In [22]:
```
"""
Parameters:
d: dimension of the sphere

Returns:
random point from boundary of d-sphere
"""
def sample_unit_sphere(d):
    ######################################################################
##########
    # TODO: Generate a point from the boundary of a d-sphere
         #
    # Refer to the link provided in the question sheet
         #
    ######################################################################
##########

    point = np.random.randn(d,1)
    point = np.array([point[0][0],point[1][0]])
    point = point/np.linalg.norm(point)

    ######################################################################
##########
    #                              END OF YOUR CODE
         #
    ######################################################################
##########
    return point
```

In [23]:
```
## Run this to check that your function is indeed sampling from the unit
 2-sphere
# You should see a circle
fig, ax = plt.subplots(1, figsize=(8,8))
ax.set_xlim(-1.1,1.1)
ax.set_ylim(-1.1,1.1)
for i in range(200):
    point = sample_unit_sphere(2)
    ax.scatter(point[0], point[1])
plt.show()
```

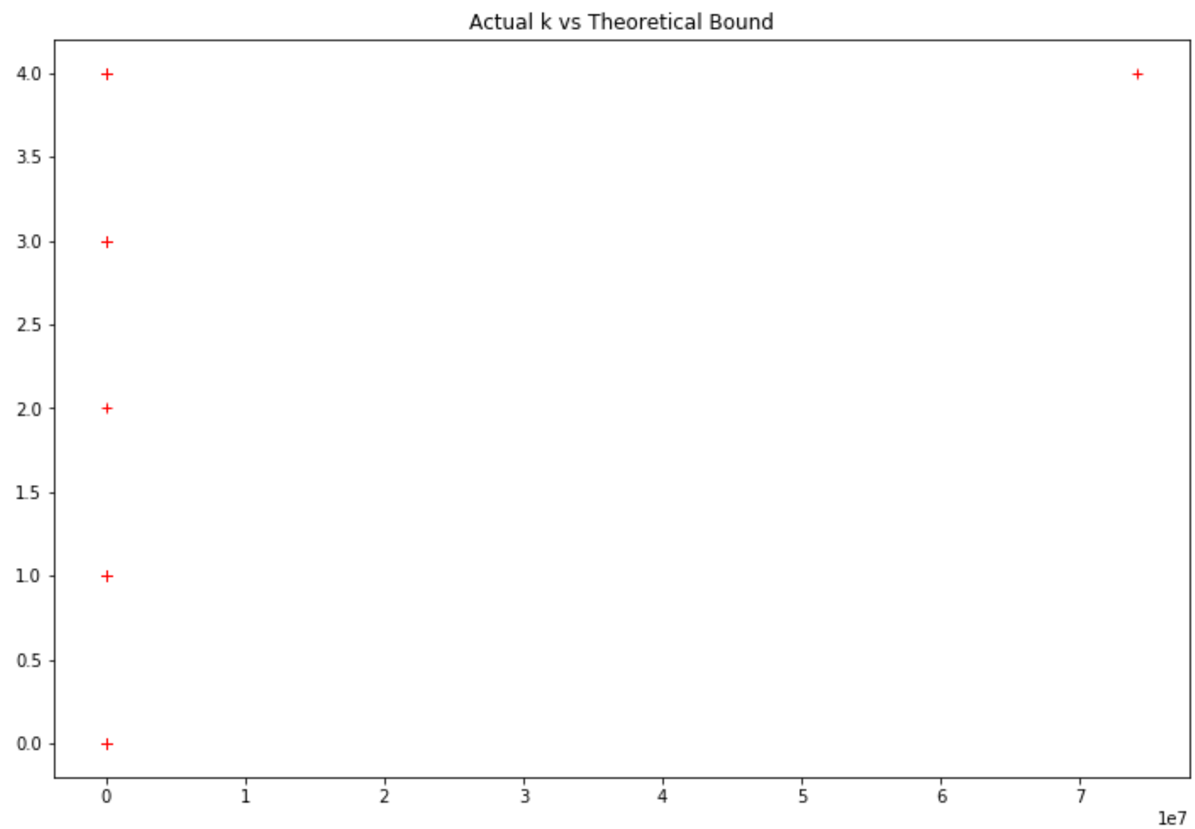


In [24]:
```
########################################################################
######
# TODO: Complete Problem 1 Q3(d) by sampling theta^{(0)} from the unit-c
ircle#
# at each round and running the std perceptron
     #
# Hint: Save the first 100 results in a vector for plotting later
     #
########################################################################
######
k = np.zeros(10000)
theta = np.zeros((100,2))
gamma = np.zeros(10000)
theoretical_upper_bound = np.zeros(100)
for i in range(10000):
    theta_init = sample_unit_sphere(2)
    print('=================Iter %s==================' % (i+1))
    print('Sample point from unit circle = %s' % theta_init)
    if(i<100):
        k[i],theta[i],gamma[i] = std_perceptron(X, y, theta_init)
        theoretical_upper_bound[i] = 1/pow(gamma[i],2)
        print('k = %s, theta = [%.3f,%.3f], gamma=%.3f' % (k[i], theta[i
][0], theta[i][1], gamma[i]))
    else:
        k[i],theta,gamma[i] = std_perceptron(X, y, theta_init)
        print('k = %s, theta = [%.3f,%.3f], gamma=%.3f' % (k[i], theta[0
], theta[1], gamma[i]))



########################################################################
######
#                                END OF YOUR CODE
     #
########################################################################
######
```

```
==================Iter 1==================
Sample point from unit circle = [-0.82021535  0.57205487]
k = 3.0, theta = [0.996,-0.086], gamma=0.236
==================Iter 2==================
Sample point from unit circle = [ 0.80547874 -0.59262467]
k = 1.0, theta = [0.958,0.286], gamma=0.275
==================Iter 3==================
Sample point from unit circle = [ 0.34508442 -0.93857165]
k = 1.0, theta = [1.000,-0.002], gamma=0.316
==================Iter 4==================
Sample point from unit circle = [ 0.7983284   0.60222236]
k = 1.0, theta = [0.984,-0.177], gamma=0.145
==================Iter 5==================
Sample point from unit circle = [-0.50022848 -0.86589345]
k = 3.0, theta = [0.983,0.182], gamma=0.375
==================Iter 6==================
Sample point from unit circle = [ 0.9963224   0.08568362]
k = 0.0, theta = [0.996,0.086], gamma=0.399
==================Iter 7==================
Sample point from unit circle = [-0.87314514  0.48746031]
k = 3.0, theta = [0.978,-0.208], gamma=0.115
==================Iter 8==================
Sample point from unit circle = [-0.8734605   0.48689501]
k = 3.0, theta = [0.978,-0.209], gamma=0.114
==================Iter 9==================
Sample point from unit circle = [ 0.14835785  0.98893374]
k = 1.0, theta = [0.960,0.281], gamma=0.279
==================Iter 10==================
Sample point from unit circle = [-0.89775206  0.44050111]
k = 3.0, theta = [0.961,-0.278], gamma=0.043
==================Iter 11==================
Sample point from unit circle = [ 0.11236829  0.99366663]
k = 1.0, theta = [0.954,0.299], gamma=0.262
==================Iter 12==================
Sample point from unit circle = [-0.69141075  0.72246189]
k = 3.0, theta = [0.995,0.096], gamma=0.408
==================Iter 13==================
Sample point from unit circle = [-0.17687807  0.98423277]
k = 1.0, theta = [0.901,0.434], gamma=0.119
==================Iter 14==================
Sample point from unit circle = [-0.9847546  -0.17394935]
k = 2.0, theta = [0.911,0.413], gamma=0.142
==================Iter 15==================
Sample point from unit circle = [-0.20017038  0.9797611 ]
k = 1.0, theta = [0.896,0.444], gamma=0.108
==================Iter 16==================
Sample point from unit circle = [-0.80523047 -0.59296196]
k = 4.0, theta = [0.959,-0.283], gamma=0.037
==================Iter 17==================
Sample point from unit circle = [-0.76839986 -0.63997004]
k = 4.0, theta = [0.951,-0.310], gamma=0.010
==================Iter 18==================
Sample point from unit circle = [ 0.44046826  0.89776818]
k = 1.0, theta = [0.991,0.131], gamma=0.423
==================Iter 19==================
Sample point from unit circle = [ 0.49726613  0.86759806]
k = 1.0, theta = [0.995,0.099], gamma=0.411
```

```
==================Iter 9995==================
Sample point from unit circle = [ 0.87111095  0.49108625]
k = 0.0, theta = [0.871,0.491], gamma=0.055
==================Iter 9996==================
Sample point from unit circle = [-0.55664833 -0.83074824]
k = 3.0, theta = [0.972,0.236], gamma=0.323
==================Iter 9997==================
Sample point from unit circle = [-0.62196517  0.78304491]
k = 3.0, theta = [0.991,0.134], gamma=0.421
==================Iter 9998==================
Sample point from unit circle = [-0.36178698 -0.93226079]
k = 3.0, theta = [0.996,0.091], gamma=0.403
==================Iter 9999==================
Sample point from unit circle = [-0.99840732 -0.05641647]
k = 4.0, theta = [0.978,0.208], gamma=0.351
==================Iter 10000==================
Sample point from unit circle = [-0.92891551 -0.37029174]
k = 4.0, theta = [0.993,-0.116], gamma=0.207
```

In [25]:
```
fig_prob1q3d, ax = plt.subplots(1, figsize=(12,8))
ax.set_title(r'Actual k vs Theoretical Bound')
########################################################################
######
# TODO: Problem 1 Q3(d) - Plotting
     #
########################################################################
######

for i in range(100):
    ax.plot(theoretical_upper_bound[i],k[i],'r+')

########################################################################
######
#                              END OF YOUR CODE
     #
########################################################################
######
plt.savefig("problem1_q3d", dpi=300, bbox_inches='tight') # Exports the
 figure
plt.show()
```



In [26]:
```
print("Average number of updates for 10,000 updates is:", np.average(k))
print("Average gamma for 10,000 updates is:", np.average(gamma))
```

```
Average number of updates for 10,000 updates is: 1.8454
Average gamma for 10,000 updates is: 0.241698538882
```

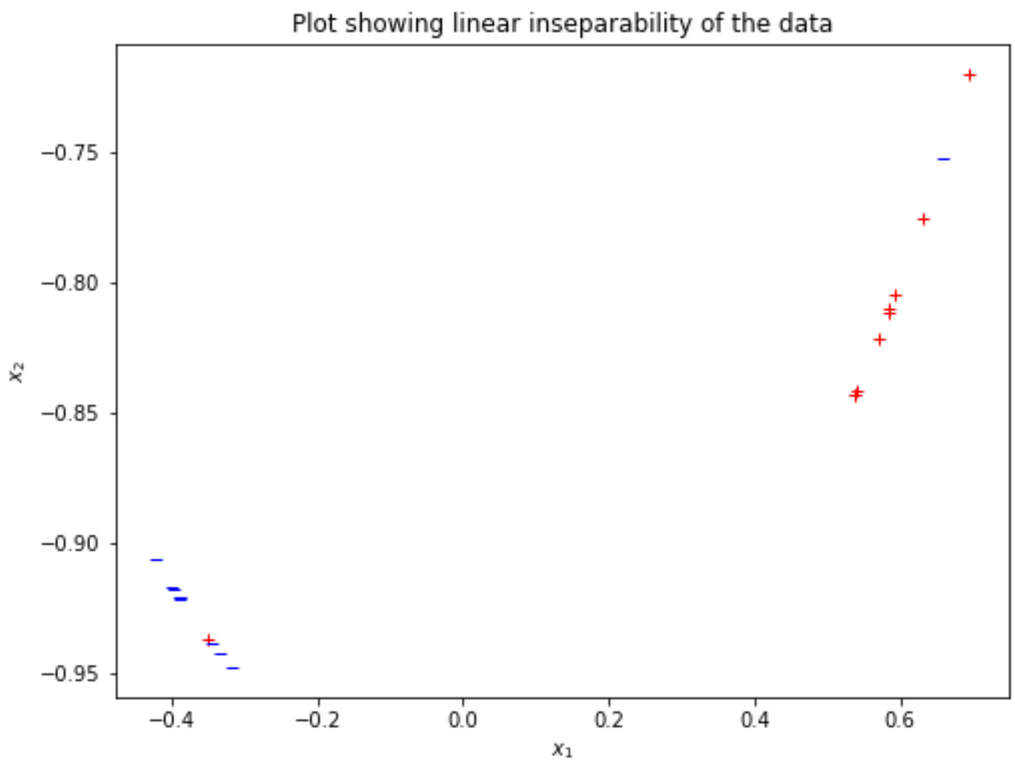
Problem 1 Q4 - Linear Inseparability

In [27]:
```
########################################################################
######
# TODO: Modify the dataset here
     #
########################################################################
######
y[0] = -1
y[2] = 1
########################################################################
######
#                                END OF YOUR CODE
     #
########################################################################
######
```

```
/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:4: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
  after removing the cwd from sys.path.
/usr/local/lib/python3.6/site-packages/ipykernel_launcher.py:5: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/indexing.html#indexing-view-versus-copy
  """
```

In [28]:

```
## Plotting
fig_prob1q4, ax = plt.subplots(1, figsize=(8,6))
y_marker = {1: 'r+', -1: 'b_'}
ax.set_title(r'Plot showing linear inseparability of the data')
ax.set_xlabel(r'$x_1$')
ax.set_ylabel(r'$x_2$')
########################################################################
######
# TODO: Plot the modified data
     #
########################################################################
######
for i in range(n):
    if(y[i]<0):
        ax.plot(x1[i],x2[i],'b_', label="Negative")
    else:
        ax.plot(x1[i],x2[i],'r+', label="Positive")

########################################################################
######
#                               END OF YOUR CODE
     #
########################################################################
######

plt.savefig("problem1_q4", dpi=300, bbox_inches='tight') # Exports the f
igure
plt.show()
```



In [32]:
```
from cvxopt import matrix, solvers
# Try running the perceptron and SVM solver on the modified dataset
# (note that you can interrupt kernel in menu bar in iPython notebook)
###########################################################################
#######
# TODO: Run std. perceptron and SVM, state your findings in the report
     #
###########################################################################
######

###########################################################################
#######
# SVM Solution
###########################################################################
######
P_new = np.eye(p,p)
q_new = np.zeros(p)
G_new = np.zeros((n,p))
h_new = -1*np.ones(n)
for i in range(n):
    for j in range(p):
        if(j==0):
            G_new[i,j] = (-1*y[i]*x1[i])
        else:
            G_new[i,j] = (-1*y[i]*x2[i])

#theta_star_modified = quadprog_solve_qp(P_new, q_new, G=G_new, h=h_new)

#theta_star_modified = solvers.qp(matrix(P_new), matrix(q_new), matrix(G
_new), matrix(h_new))


###########################################################################
#######
# Perceptron Solution
###########################################################################
######
theta_zero = np.zeros(p)
k, theta, gamma = std_perceptron(X, y, theta_zero)
print("Iterations taken to classify current non-separable data:",k)

###########################################################################
######
#                                END OF YOUR CODE
     #
###########################################################################
######
```

```
Max iterations exceeded; data is non-separable!
Iterations taken to classify current non-separable data: 1003
```

Problem 2

Preamble

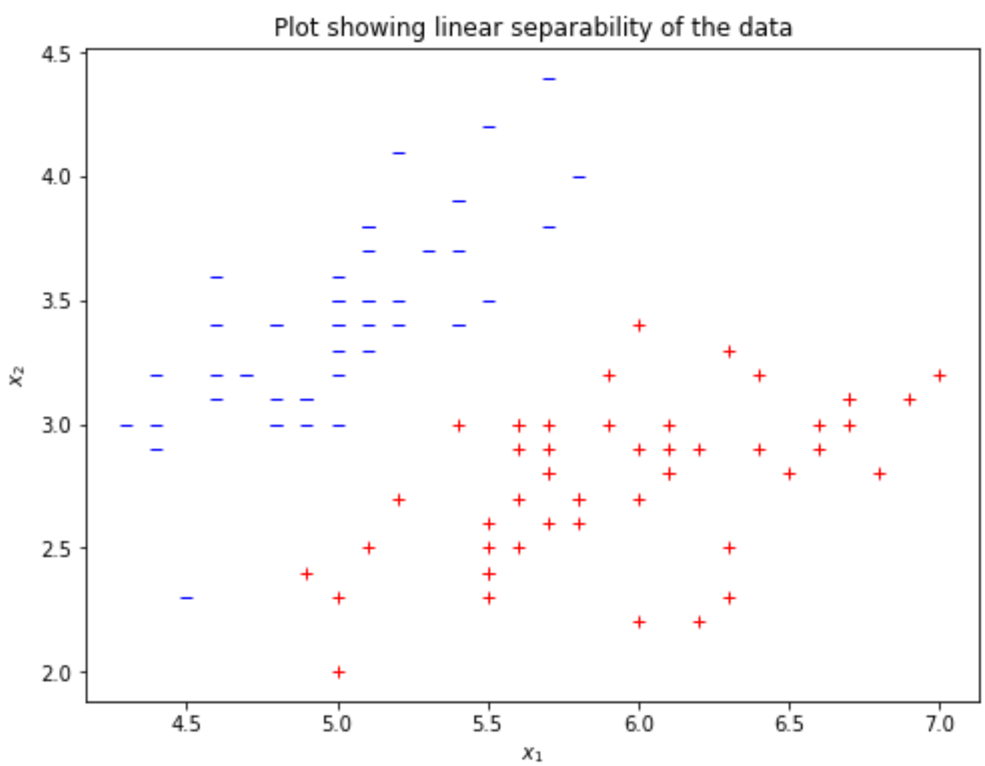
In [33]:
```
# Replace with file path of iris1.csv if not in the same folder as this
 notebook
prob2_fp = 'iris1.csv'

# Import data
prob2_df = pd.read_csv(prob2_fp, header=None, names=['x1','x2','y'])

# Set up problem parameters and vectors
n, p, x1, x2, y = prob2_df.shape[0], prob2_df.shape[1] - 1, prob2_df['x
1'], prob2_df['x2'], prob2_df['y']
X = prob2_df[['x1','x2']]

# Preview the first 5 rows
prob2_df.head(5)
```

Out[33]:

	x1	x2	y
0	5.1	3.5	-1
1	4.9	3.0	-1
2	4.7	3.2	-1
3	4.6	3.1	-1
4	5.0	3.6	-1

Problem 2 Q1

In [34]:
```
fig_prob2q1, ax = plt.subplots(1, figsize=(8,6))
y_marker = {1: 'r+', -1: 'b_'}
ax.set_title(r'Plot showing linear separability of the data')
ax.set_xlabel(r'$x_1$')
ax.set_ylabel(r'$x_2$')
########################################################################
######
# TODO: Plot the labeled data by looking up the y_marker dictionary, ref
er   #
# to: https://matplotlib.org/devdocs/api/_as_gen/matplotlib.pyplot.plot.
html #
########################################################################
######

for i in range(n):
    if(y[i]<0):
        ax.plot(x1[i],x2[i],'b_', label="Negative")
    else:
        ax.plot(x1[i],x2[i],'r+', label="Positive")

########################################################################
######
#                              END OF YOUR CODE
     #
########################################################################
######

plt.savefig("problem2_q1", dpi=300, bbox_inches='tight') # Exports the f
igure
plt.show()
```



Problem 2 Q2

In [35]:
```
from cvxopt import matrix, solvers

############################################################################
######
# TODO: Similar to Problem 1 Q3,
     #
# but amend the matrices to accommodate for the offset
     #
############################################################################
######
P = np.eye(p+1,p+1)
q = np.zeros(p+1)
G = np.zeros((n,p+1))
h = -1*np.ones(n)
for i in range(n):
    for j in range(p+1):
        if(j==0):
            G[i,j] = (-1*y[i]*x1[i])
        elif(j==1):
            G[i,j] = (-1*y[i]*x2[i])
        else:
            G[i,j] = (-1*y[i])

theta_hardsvm = quadprog_solve_qp(P, q, G=G, h=h)
theta_0 = theta_hardsvm[2]
theta_hardsvm = np.array([theta_hardsvm[0],theta_hardsvm[1]])
#sol = solvers.qp(matrix(P),matrix(q),matrix(G),matrix(h))
print('Problem 2 Q2 \n=================================')
print('theta_0*: %s' % theta_0)
print('theta^*: %s' % theta_hardsvm)
print('Optimal objective value is: %s' % ((pow(np.linalg.norm(theta_hard
svm),2))/2))

############################################################################
######
#                               END OF YOUR CODE
     #
############################################################################
######
```

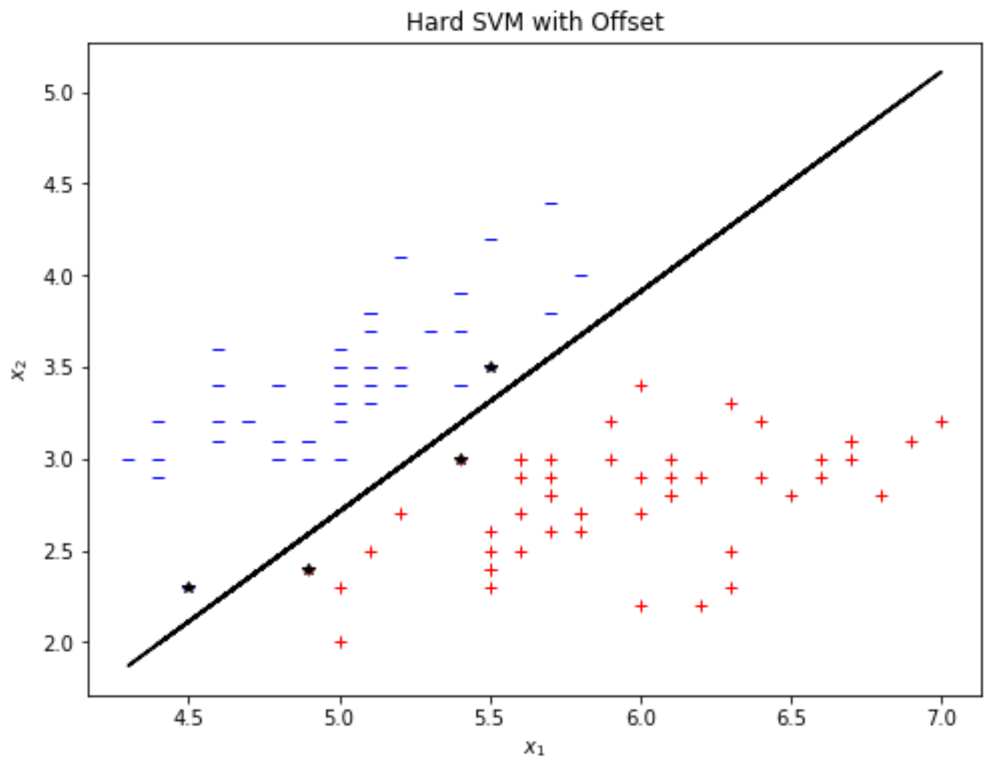
```
Problem 2 Q2
=================================
theta_0*: -17.3157894737
theta^*: [ 6.31578947 -5.26315789]
Optimal objective value is: 33.7950138504
```

Problem 2 Q3

In [36]:
```
## First detect the support vectors
## Collect their indices into the vector SV
## Note that the indices here are in {0,1,...,99}. In your report, give
 the indices in {1,2,...,100}
SV = []
for i in range(n):
    ####################################################################
##########
    # TODO: Detect support vectors and append their indices into SV
         #
    ####################################################################
##########
    value = (theta_hardsvm[0]*x1[i]+theta_hardsvm[1]*x2[i])+theta_0
    value = round(value,1)
    if(value == 1.0 or value == -1.0):
        SV.append(i)
    ####################################################################
##########
    #                              END OF YOUR CODE
         #
    ####################################################################
##########
```

In [37]:
```
fig_prob2q3, ax = plt.subplots(1, figsize=(8,6))
y_marker = {1: 'r+', -1: 'b_'}
ax.set_title(r'Hard SVM with Offset')
ax.set_xlabel(r'$x_1$')
ax.set_ylabel(r'$x_2$')
########################################################################
######
# TODO: Plot the original data
     #
########################################################################
######

for i in range(n):
    if(y[i]<0):
        ax.plot(x1[i],x2[i],'b_', label="Negative")
    else:
        ax.plot(x1[i],x2[i],'r+', label="Positive")

########################################################################
######
#                                END OF YOUR CODE
     #
########################################################################
######




########################################################################
######
# TODO: Plot the decision boundary
     #
########################################################################
######

f = lambda x1 : a*x1 +b
a = -theta_hardsvm[0]/theta_hardsvm[1]
b = -theta_0/theta_hardsvm[1]
# draw the line in blac'k'
ax.plot(x1,f(x1),'k', label="Separator")

########################################################################
######
#                                END OF YOUR CODE
     #
########################################################################
######





########################################################################
######
# TODO: Indicate the support vectors
     #
```

```
##########################################################################
######

for i in range(len(SV)):
    ax.plot(x1[SV[i]],x2[SV[i]],'k*', label="Support Vector")
##########################################################################
######
#                              END OF YOUR CODE
     #
##########################################################################
######

plt.savefig("problem2_q3", dpi=300, bbox_inches='tight') # Exports the f
igure
plt.show()
```



Problem 2 Q4

In [39]:
```
##########################################################################
######
# TODO: Write the problem in matrix notation and solve with optimization
     #
# software (e.g. quadprog)
     #
##########################################################################
######

"""
Note: You might need to perturb the
objective matrix along the diagonal by positve eps,
to ensure that it is positive definite
"""
eps = 3e-8
P = np.zeros((n,p))
q = -np.ones(n)
G = -1*np.eye(n)
h = np.zeros(n)
b = 0
A = np.reshape(np.array([y]),(1,100))

for i in range(n):
    for j in range(p):
        if(j==0):
            P[i,j] = (y[i]*x1[i])
        else:
            P[i,j] = (y[i]*x2[i])

P = np.matmul(P,np.transpose(P)) + eps*np.eye(n,n)

alpha_star = quadprog_solve_qp(P=P, q=q, G=G, h=h, A=A, b=b)
objective = -0.5*np.matmul(np.matmul(np.transpose(alpha_star),P),alpha_s
tar)+np.sum(alpha_star)
print("Objective value is:",objective)
indices=[]
for i in range(n):
    if(alpha_star[i]>10**-6):
        indices.append(i)
        print("Alpha from quadprog is:",alpha_star[i], " at index:", i)

#alpha_alt = sol['z']
# for i in range(n):
#     if(alpha_alt[i]>10**-6):
#             print("Alpha from CVXOPT is:",alpha_alt[i], " at index:",
 i)



##########################################################################
######
#                                END OF YOUR CODE
     #
##########################################################################
######
```

```
Objective value is: 33.7949973533
Alpha from quadprog is: 15.9002661526  at index: 36
Alpha from quadprog is: 17.8947302452  at index: 41
Alpha from quadprog is: 16.3988888621  at index: 57
Alpha from quadprog is: 17.3961076312  at index: 84
```

Problem 2 Q5

In [41]:

```
import random
###########################################################################
######
# TODO: Do your calculations here
     #
###########################################################################
######

theta_kkt = np.zeros((n,p))
for i in range(n):
    for j in range(p):
        if(j==0):
            theta_kkt[i,j] = (y[i]*x1[i])
        else:
            theta_kkt[i,j] = (y[i]*x2[i])

theta_kkt = np.matmul(np.transpose(alpha_star),theta_kkt)

print("First sum which gives theta's value is:",theta_kkt)

index = random.choice(indices)
theta_0_kkt = y[index]-(theta_kkt[0]*x1[index]+theta_kkt[1]*x2[index])
print("Index is", index, "and the second sum which gives theta_0's value
 is:",theta_0_kkt)
###########################################################################
######
#                               END OF YOUR CODE
     #
###########################################################################
######
```

```
First sum which gives theta's value is: [ 6.31578606 -5.26315523]
Index is 84 and the second sum which gives theta_0's value is: -17.3157
790286
```

Problem 3

Preamble

In [9]:
```
# Replace with file path of iris1.csv if not in the same folder as this
 notebook
prob3_fp = 'iris2.csv'

# Import data
prob3_df = pd.read_csv(prob3_fp, header=None, names=['x1','x2','x3','x4'
,'y'])

# Set up problem parameters and vectors
n, p, X, y = prob3_df.shape[0], prob3_df.shape[1] - 1, prob3_df[['x1','x
2','x3','x4']], prob3_df['y']
x1,x2,x3,x4 = prob3_df['x1'],prob3_df['x2'],prob3_df['x3'],prob3_df['x4'
]

normalized_X = np.zeros((n,p+1))

for i in range(n):
    norm = 1
    normalized_X[i,0] = x1[i]/norm
    normalized_X[i,1] = x2[i]/norm
    normalized_X[i,2] = x3[i]/norm
    normalized_X[i,3] = x4[i]/norm
    normalized_X[i,4] = 1

# Preview the first 5 rows
prob3_df.head(5)
```

Out[9]:

	x1	x2	x3	x4	y
0	7.0	3.2	4.7	1.4	-1
1	6.4	3.2	4.5	1.5	-1
2	6.9	3.1	4.9	1.5	-1
3	5.5	2.3	4.0	1.3	-1
4	6.5	2.8	4.6	1.5	-1

In [27]:
```
def train_SVM(C,X,y):
    eps = 3e-8
    n = len(X)
    p = 4
    P = np.zeros((n+p+1,n+p+1))

    for i in range(p):
        P[i,i] = 1-eps
        P = P + eps*np.eye(n+p+1,n+p+1)

    C = C
    q = np.zeros(n+p+1)

    for i in range(n):
        q[i+p+1] = C

    G = np.zeros((n+n,n+p+1))
    h = np.zeros(n+n)

    for i in range(n):
        h[i] = 1

    for i in range(n):
        for j in range(p+1):
            G[i,j] = y[i]*X[i,j]

    for i in range(n):
        G[i,p+i+1] = 1

    for i in range(n):
        G[i+n,i+p+1] = 1


    G = -G
    h = -h

    theta = quadprog_solve_qp(P, q, G,h)
    return theta
```

Problem 3 Q1 - Soft SVM

In [28]:

```
"""
Note: You might need to perturb the
objective matrix along the diagonal by positve eps,
to ensure that it is positive definite
"""

C = 100
theta_softSVM_original = train_SVM(C,normalized_X,y)

epsilon = np.zeros(n)
for i in range(n):
    epsilon[i] = theta_softSVM_original[i+p+1]

theta_0 = theta_softSVM_original[4]
theta_softSVM_mult = np.array([theta_softSVM_original[0],theta_softSVM_o
riginal[1],theta_softSVM_original[2],theta_softSVM_original[3],theta_sof
tSVM_original[4]])
theta_softSVM = np.array([theta_softSVM_original[0],theta_softSVM_origin
al[1],theta_softSVM_original[2],theta_softSVM_original[3]])

objective_value = 0.5*(pow(np.linalg.norm(theta_softSVM),2)) + C*np.sum(
epsilon)

print('Problem 3 Q1 \n=================================')
print('theta_0*: %s' % theta_0)
print('theta^*: %s' % theta_softSVM)
print('Optimal value is:',objective_value)

misclassified = []

for i in range(n):
    if(y[i]*(np.matmul(np.transpose(theta_softSVM_mult),normalized_X[i
]))<0):
        misclassified.append(i)

print("Misclassified points are",len(misclassified),"at:",str(misclassif
ied))
```

```
Problem 3 Q1
=================================
theta_0*: -20.4130434783
theta^*: [ -1.84782609  -3.26086957   4.67391304  10.86956522]
Optimal value is: 654.194234405
Misclassified points are 3 at: [20, 33, 83]
```

Problem 3 Q2 - Cross-validation

In [32]:
```
def splitX(X,index):
    X_train =  np.zeros((90,5))
    X_test = np.zeros((10,5))
    to_split =  index*10
    till = to_split+10
    k1=0
    k2=0
    for i in range(100):
        if(i>=to_split and i<till):
            X_test[k2] = X[i]
            k2+=1
        else:
            X_train[k1] = X[i]
            k1+=1
    return X_train,X_test
def splitY(Y,index):
    Y_train =  np.zeros((90,1))
    Y_test = np.zeros((10,1))
    to_split =  index*10
    till = to_split+10
    k1=0
    k2=0
    for i in range(100):
        if(i>=to_split and i<till):
            Y_test[k2] = Y[i]
            k2+=1
        else:
            Y_train[k1] = Y[i]
            k1+=1
    return Y_train,Y_test

def train_SVM_error(C,X_split_train,X_split_test,Y_split_train,Y_split_t
est):
    eps = 3e-8
    n = len(X_split_train)
    p = 4
    P = np.zeros((n+p+1,n+p+1))

    for i in range(p):
        P[i,i] = 1-eps
        P = P + eps*np.eye(n+p+1,n+p+1)

    C = C
    q = np.zeros(n+p+1)

    for i in range(n):
        q[i+p+1] = C

    G = np.zeros((n+n,n+p+1))
    h = np.zeros(n+n)

    for i in range(n):
        h[i] = 1

    for i in range(n):
        for j in range(p+1):
```

```
            G[i,j] = Y_split_train[i]*X_split_train[i,j]

    for i in range(n):
        G[i,p+i+1] = 1

    for i in range(n):
        G[i+n,i+p+1] = 1


    G = -G
    h = -h

    theta_softSVM_original = quadprog_solve_qp(P, q, G,h)
    theta_softSVM_mult = np.array([theta_softSVM_original[0],theta_softS
VM_original[1],theta_softSVM_original[2],theta_softSVM_original[3],theta
_softSVM_original[4]])
    count = 0

    for i in range(len(Y_split_test)):
        if(Y_split_test[i]*(np.matmul(np.transpose(theta_softSVM_mult),X
_split_test[i]))<0):
            count+=1

    return count;

def ten_fold_cross_validation(C,X,y):

    error = 0

    for i in range(10):
        X_split_train,X_split_test = splitX(X,i)
        Y_split_train,Y_split_test = splitY(y,i)
        error+=train_SVM_error(C,X_split_train,X_split_test,Y_split_trai
n,Y_split_test)

    return error/10;
```

In [33]:

```
C = [1,100,10000]
av_test_error_by_C = []
for c in C:
    test_errors = []
    test_errors.append(ten_fold_cross_validation(c,normalized_X,y))
    av_test_error = np.mean(test_errors)
    print('C = %s, Av. test error = %s' % (c, av_test_error))
    av_test_error_by_C.append(av_test_error)

print('Optimal C = %s' % C[np.argmin(av_test_error_by_C)])
```

```
C = 1, Av. test error = 0.4
C = 100, Av. test error = 0.6
C = 10000, Av. test error = 0.7
Optimal C = 1
```