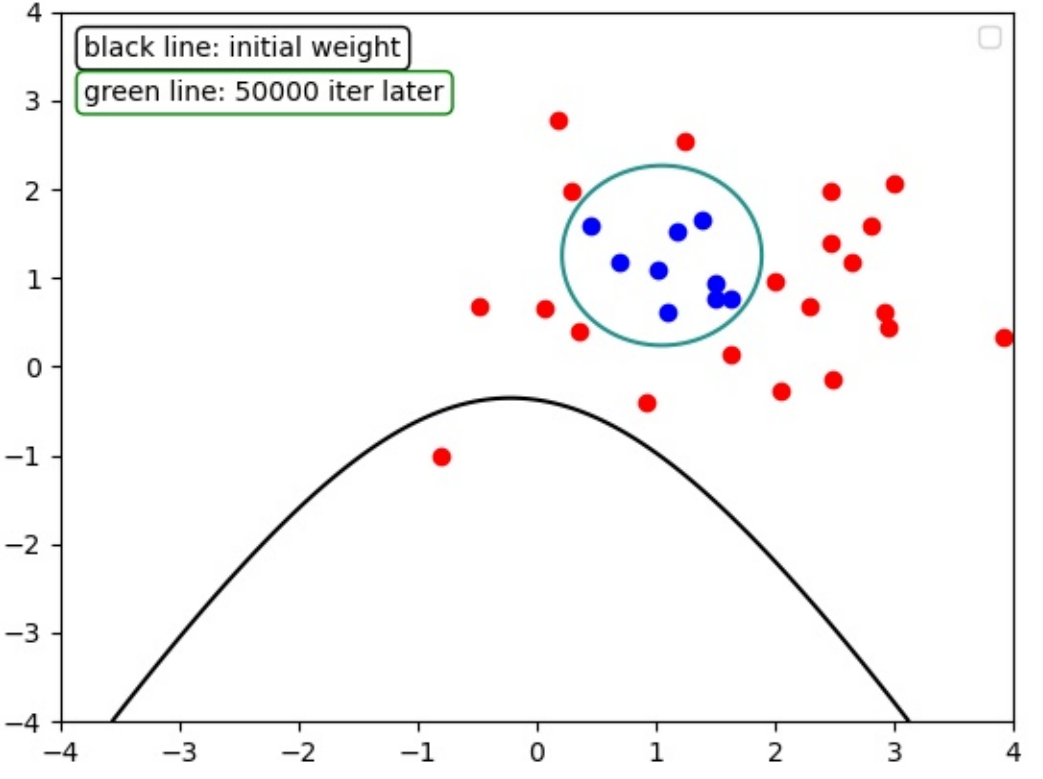
```
import numpy as np
import matplotlib.pyplot as plt

N=30
np.random.seed(0)
X = np.random.randn(2,N)
y = np.sign(X[0 ,:]**2+X[1 ,:]**2 -0.7)
theta = 0.5
c, s = np.cos(theta), np.sin(theta)
X = np.array ([[c, -s], [s, c]])@X
X = X + np.array ([[1] ,[1]])

## initialize weight
Theta = np.random.randn(5)

## make new data
X_kernel = []

for p in X.T:
    row = [1]
    row += [p[0],p[0]**2,p[1],p[1]**2]
    X_kernel.append(row)

X_kernel = np.array(X_kernel)

## draw initial bound
xx = np.linspace(-4, 4, 1024)
yy = np.linspace(-4, 4, 1024)
xx , yy = np.meshgrid(xx , yy)
w = Theta[:]
Z = w[0] + (w[1] * xx + w[2] * xx**2) + (w[3] * yy + w[4] * yy**2)
plt.contour(xx , yy , Z, 0, colors='k',)


## SGD with SVM
def calculate_loss(x,y,theta):
    return np.average(np.max( 1-y*(x@theta) , 0)) + lamb*np.sum(theta**2)

max_iter = 50000
lr = 0.01
lamb = 0.0001

## store loss each iteration
loss = []


## training
for i in range(max_iter):
    # choose random 1 data
    index = np.random.randint(0,N)

    # just temporary instance
    val = y[index]*X_kernel[index]@Theta
```

```
    if val >=1 : gradient = 0
    else: gradient = -y[index]*X_kernel[index]

    # regularization part
    gradient += 2*lamb*Theta

    loss.append(calculate_loss(X_kernel,y,Theta))
    Theta -= lr*gradient

w = Theta[:]
Z = w[0] + (w[1] * xx + w[2] * xx**2) + (w[3] * yy + w[4] * yy**2)
plt.contour(xx , yy , Z, 0, )

# plot the points
plt.scatter(X[0][y>0], X[1][y>0], c='r')
plt.scatter(X[0][y<0], X[1][y<0], c='b')

box1 = {'boxstyle': 'round',
        'ec': (0, 0, 0),
        'fc': (1, 1, 1)}
box2 = {'boxstyle': 'round',
        'ec': (0, 0.5, 0),
        'fc': (1, 1, 1)}

plt.text(-3.8,3.5, 'black line: initial weight', bbox=box1)
plt.text(-3.8,3, f'green line: {max_iter} iter later', bbox=box2)

plt.savefig('p3_visualization.png')
plt.show()
```


black line: initial weight
green line: 50000 iter later
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3
4