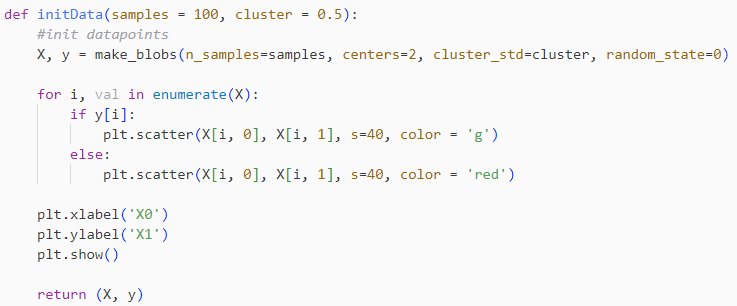
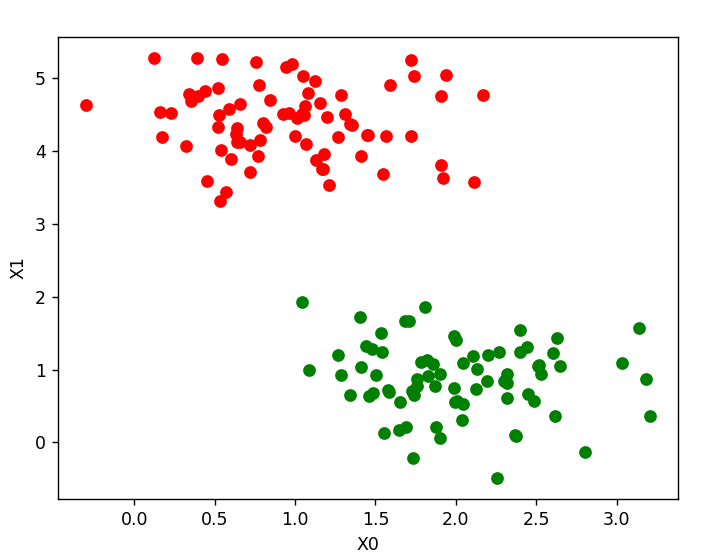
**HW8: SVM, soft margin SVM, kernel SVM**

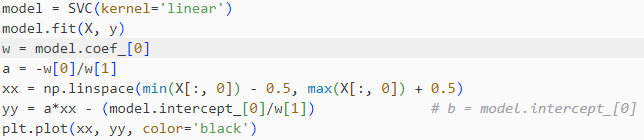
* SVM: khá giống với thuật toán Perceptron, nhưng Perceptron ta có thể tìm được vô số đường hyper-plane bất kì để phân chia 2 lớp. Còn với SVM sẽ tìm hyper-plane thỏa điều kiện margin đạt giá trị lớn nhất.
  + Khởi tạo các Datapoint phù hợp với bài toán linear:



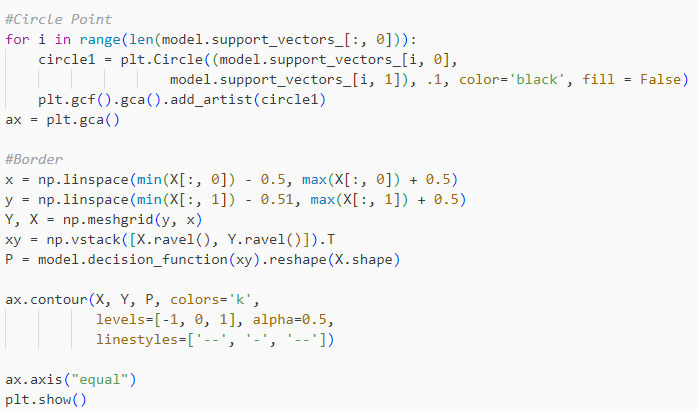
* + Với giá số mẫu là 150 và cluster = 0.5, ta được:



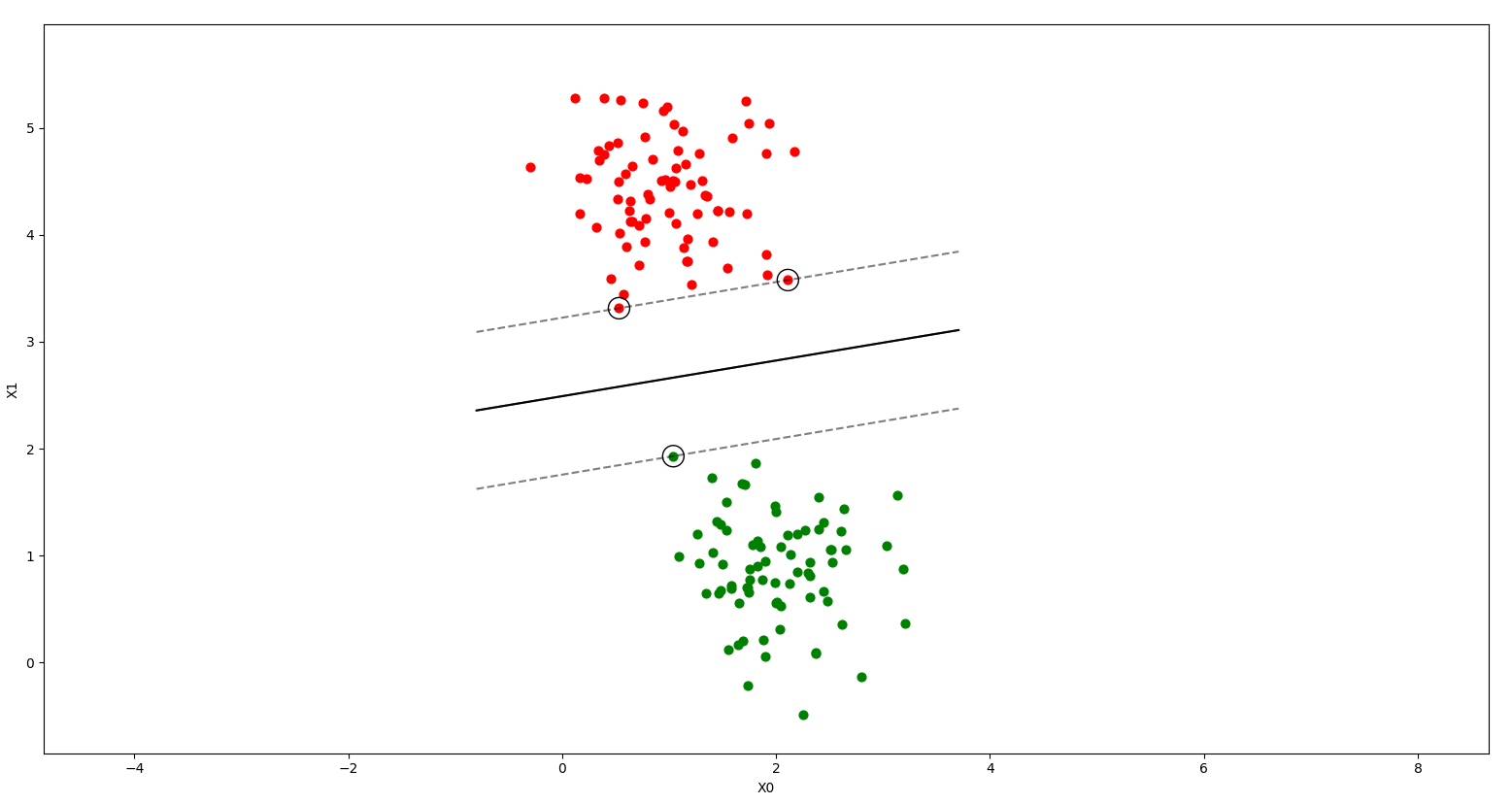
* + Sau khi có được các datapoint cần thiết, tiến hành train bằng thư viện sklearn:



* + - Sau khi thu được vecto w, tiến hành vẽ đường hyper-plane
  + Và cuối cùng là khoanh tròn các điểm và vẽ đường thẳng để xác định margin:



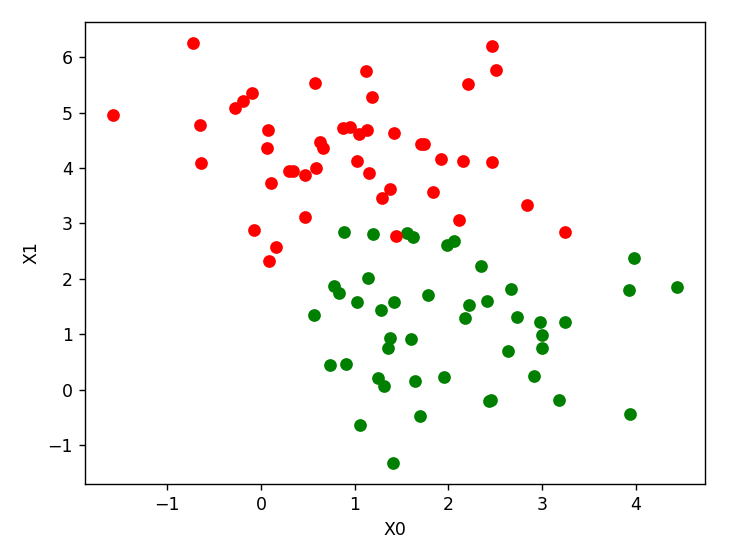
* + Kết quả cuối cùng :



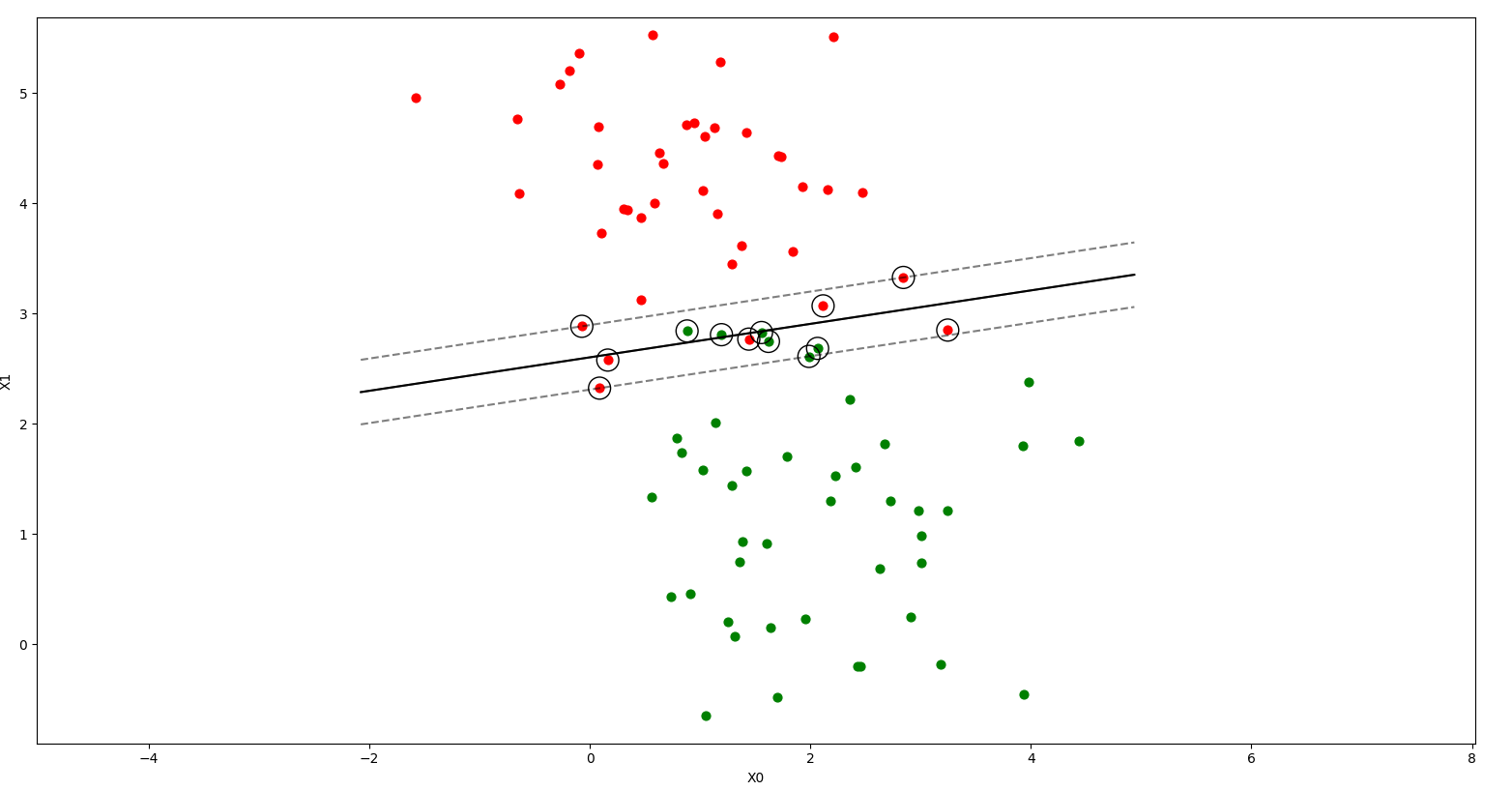
* Soft SVM: tương tự với SVM, tuy nhiên soft SVM có thể làm việc được với những dữ liệu gần linear.
  + Tương tự với SVM, đầu tiên cũng đi khởi tạo các datapoint nhưng ta sẽ thay cluster = 1 để tập datapoint chỉ gần linear và lựa chọn C = 55:



* + Tập dữ liệu lúc này:

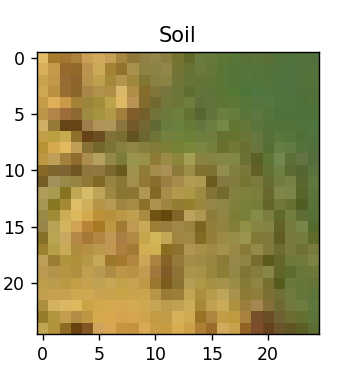
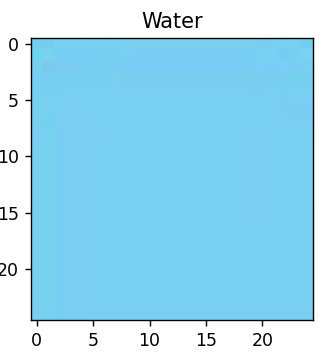


* + Kết quả sau khi chạy:



* Áp dụng bài toàn SVM vào xử lý ảnh: Cho bức ảnh dưới đây:

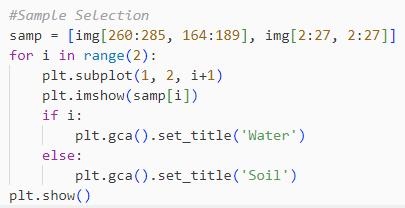


Sử dụng SVM với 2 mẫu là (đất liền) và (biển) để tách ra 2 map gồm có 1 map chỉ có đất liền, 1 map chỉ có biển.

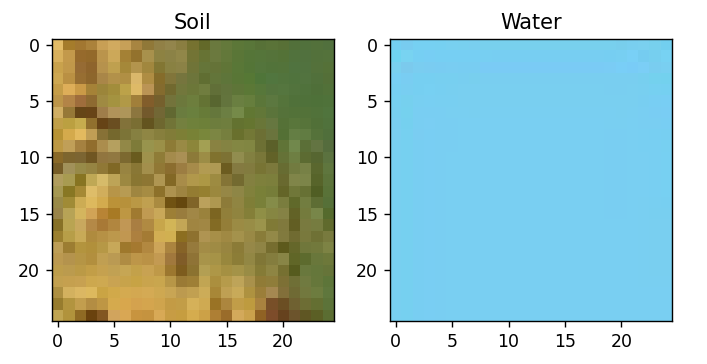
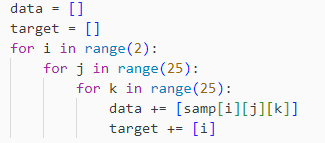
* + Đầu tiên, ta sẽ load bức ảnh và lưu giá trị vào biến img:



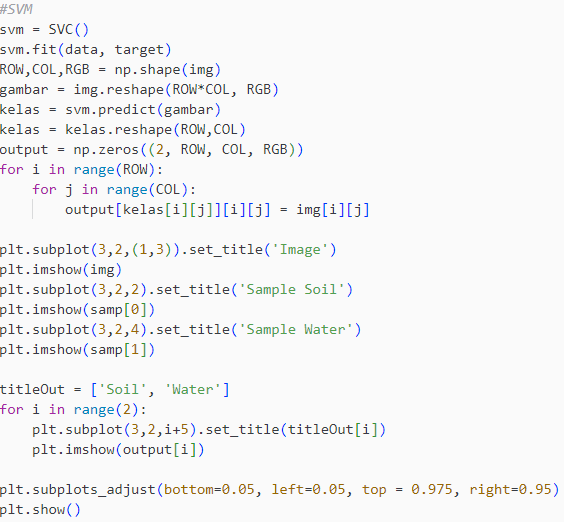
* + Tiếp theo lựa chọn 2 mẫu đất liền và biển và lưu vào list samp:

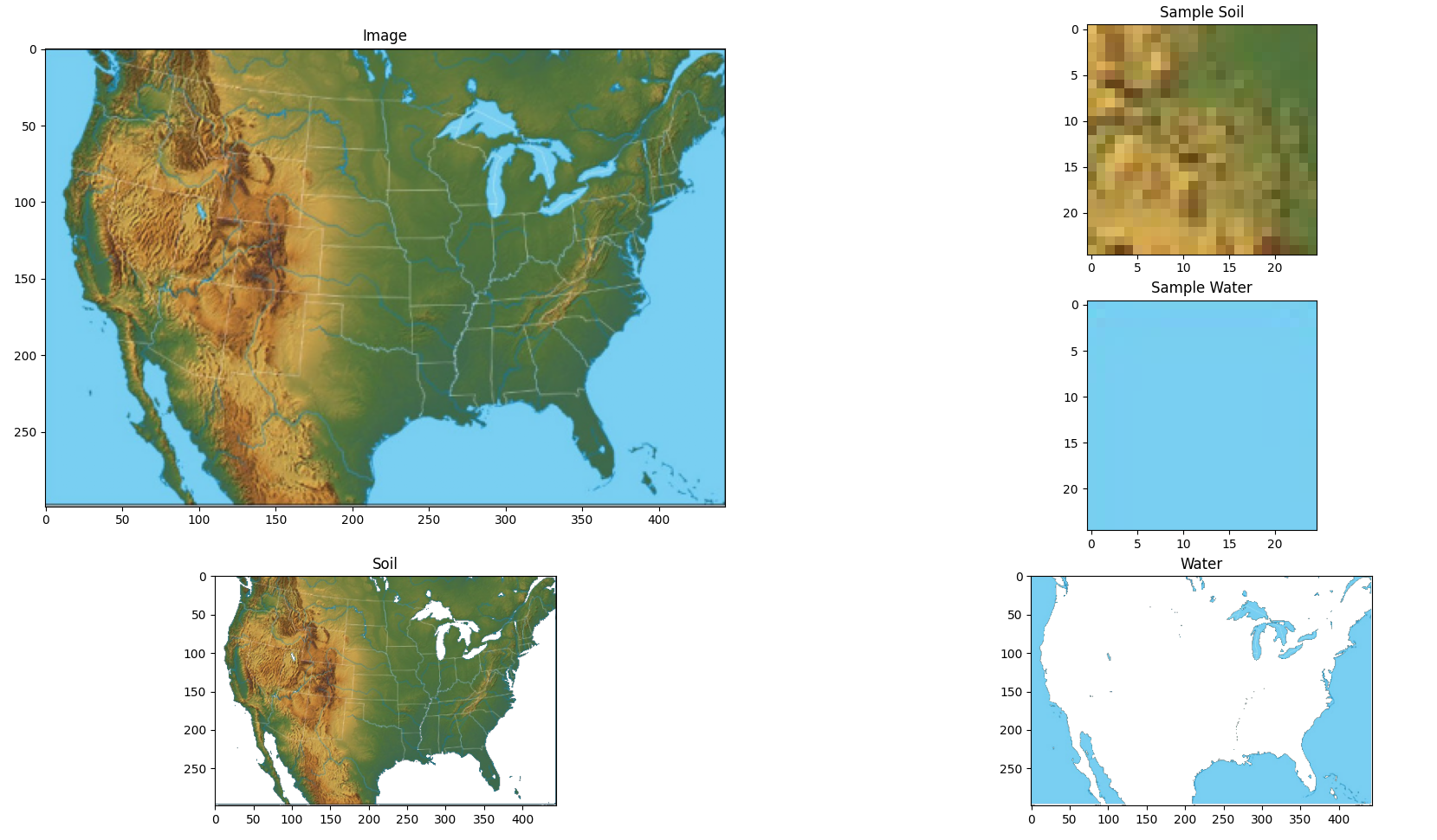


* + Các gái trị train và mẫu sẽ được để ở list data và target:



* + Tiến hành train SVM và hiển thị kết quả thu được sau khi train:





* Full code:

*#hard margin SVM, soft margin SVM*

from turtle import title

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets.\_samples\_generator import make\_blobs

from sklearn.svm import SVC

def initData(samples = 100, cluster = 0.5):

*#init datapoints*

    X, y = make\_blobs(n\_samples=samples, centers=2, cluster\_std=cluster, random\_state=0)

    for i, val in enumerate(X):

        if y[i]:

            plt.scatter(X[i, 0], X[i, 1], s=40, color = 'g')

        else:

            plt.scatter(X[i, 0], X[i, 1], s=40, color = 'red')

    plt.xlabel('X0')

    plt.ylabel('X1')

    plt.show()

    return (X, y)

def SVM(X, y, type = None, C = None):

    if type == 'hard':                                  *#hard margin SVM*

        model = SVC(kernel='linear')                    *# Create hyperplane: linear*

    elif type == 'soft':                                *#soft margin SVM*

        model = SVC(kernel='linear', C = C)             *# Create hyperplane: linear*

    model.fit(X, y)

    w = model.coef\_[0]

    a = -w[0]/w[1]

    xx = np.linspace(min(X[:, 0]) - 0.5, max(X[:, 0]) + 0.5)

    yy = a\*xx - (model.intercept\_[0]/w[1])              *# b = model.intercept\_[0]*

    plt.plot(xx, yy, color='black')

    for i, val in enumerate(X):

        if y[i]:

            plt.scatter(X[i, 0], X[i, 1], s=40, color = 'g')

        else:

            plt.scatter(X[i, 0], X[i, 1], s=40, color = 'red')

    plt.xlabel('X0')

    plt.ylabel('X1')

*#Circle Point*

    for i in range(len(model.support\_vectors\_[:, 0])):

        circle1 = plt.Circle((model.support\_vectors\_[i, 0], model.support\_vectors\_[i, 1]), .1, color='black', fill = False)

        plt.gcf().gca().add\_artist(circle1)

    ax = plt.gca()

*#Border*

    x = np.linspace(min(X[:, 0]) - 0.5, max(X[:, 0]) + 0.5)

    y = np.linspace(min(X[:, 1]) - 0.51, max(X[:, 1]) + 0.5)

    Y, X = np.meshgrid(y, x)

    xy = np.vstack([X.ravel(), Y.ravel()]).T

    P = model.decision\_function(xy).reshape(X.shape)

    ax.contour(X, Y, P, colors='k',

               levels=[-1, 0, 1], alpha=0.5,

               linestyles=['--', '-', '--'])

    ax.axis("equal")

    plt.show()

def processImage():

*#Load Image*

    img = plt.imread('image.png')

    plt.imshow(img)

*#Sample Selection*

    samp = [img[260:285, 164:189], img[2:27, 2:27]]

    for i in range(2):

        plt.subplot(1, 2, i+1)

        plt.imshow(samp[i])

        if i:

            plt.gca().set\_title('Water')

        else:

            plt.gca().set\_title('Soil')

    plt.show()

    data = []

    target = []

    for i in range(2):

        for j in range(25):

            for k in range(25):

                data += [samp[i][j][k]]

                target += [i]

*#SVM*

    svm = SVC(C = 1)

    svm.fit(data, target)

    ROW,COL,RGB = np.shape(img)

    gambar = img.reshape(ROW\*COL, RGB)

    kelas = svm.predict(gambar)

    kelas = kelas.reshape(ROW,COL)

    output = np.zeros((2, ROW, COL, RGB))

    for i in range(ROW):

        for j in range(COL):

            output[kelas[i][j]][i][j] = img[i][j]

    plt.subplot(3,2,(1,3)).set\_title('Image')

    plt.imshow(img)

    plt.subplot(3,2,2).set\_title('Sample Soil')

    plt.imshow(samp[0])

    plt.subplot(3,2,4).set\_title('Sample Water')

    plt.imshow(samp[1])

    titleOut = ['Soil', 'Water']

    for i in range(2):

        plt.subplot(3,2,i+5).set\_title(titleOut[i])

        plt.imshow(output[i])

    plt.subplots\_adjust(bottom=0.05, left=0.05, top = 0.975, right=0.95)

    plt.show()

if \_\_name\_\_ == '\_\_main\_\_':

    input = input("Input = ")

    if input == 'hard':

        (X, y) = initData(150, 0.5)

        SVM(X, y, type='hard')

    elif input == 'soft':

        (X, y) = initData(90, 1)

        SVM(X, y, type='soft', C=55)

    else:

        processImage()