```
import random
import numpy as np
import copy as cp
from numpy import linalg as LA
from sklearn.svm import SVC
import matplotlib.pyplot as plt

def linreg(x,y): #function to perform linear regression
    return np.dot(np.linalg.pinv(x),y)

def find_center(point,centers):
    distances = np.array([LA.norm(point-centers[i]) for i in range(len(centers))])
    return np.argmin(distances)

def lloyd(points,centers):
    closest_centers = [find_center(points[i],centers) for i in range(len(points))]
    for i in range(len(centers)):
        cluster = [j for j in range(len(closest_centers)) if closest_centers[j] == i]
        if len(cluster) == 0:
            return "empty"
        centers[i,0] = sum([points[cluster[k],0]/len(cluster) for k in range(len(cluster))])
        centers[i,1] = sum([points[cluster[k],1]/len(cluster) for k in range(len(cluster))])
    return [sum([LA.norm(points[i]-centers[closest_centers[i]])**2 for i in range(len(points
    ))]),centers]

def rbf(points,numcenters,y,gamma):
    centers = 2 * np.random.random_sample((numcenters,2)) - 1
    phi = np.zeros([len(points),numcenters])
    min_center = 0
    min_center1 = 200
    first = True
    while min_center1 < min_center or first:
        first = False
        min_center = cp.copy(min_center1)
        test = lloyd(points,centers)
        if test == "empty":
            return ["empty","empty"]
        min_center1 = cp.copy(test[0])
        centers = cp.copy(test[1])
    for i in range(len(phi)):
        for j in range(len(phi[0])):
            phi[i,j] = np.exp((-gamma*LA.norm(points[i]-centers[j])**2))
    return [linreg(phi,y),centers]

def inerror(w,centers,gamma,points): #find the in-sample error
    error = 0
    for i in range(len(points)):
        y = np.sign(points[i,1]-points[i,0]+0.25*np.sin(np.pi*points[i,0]))
        testy = 0
        for j in range(len(w)):
            testy += w[j]*np.exp(-gamma*LA.norm(points[i]-centers[j])**2)
        if np.sign(testy) != y:
            error += 1
```

```
    #print(error)
    return error/len(points)

def outerror(w,centers,gamma,points):
    error = 0
    for i in range(len(points)):
        randx = points[i,0]
        randy = points[i,1]
        y = np.sign(randy-randx+0.25*np.sin(np.pi*randx))
        testy = 0
        for j in range(len(w)):
            testy += w[j]*np.exp(-gamma*LA.norm(np.array([randx,randy])-centers[j])**2)
        if np.sign(testy) != y:
            error += 1
    #print(error/1000)
    return error/len(points)

reg_E_in = 0
counter = 0
a16 = 0
b16 = 0
c16 = 0
d16 = 0
e16 = 0
fail = 0
kern_better = 0
kern_better2 = 0

#parameters
gamma = 1.5
gamma2 = 2
K = 9
N = 100
K2 = 12

for i in range(N):
    points = 2 * np.random.random_sample((100,2)) - 1
    y = np.array([np.sign(points[i,1]-points[i,0]+0.25*np.sin(np.pi*points[i,0])) for i in range
    (len(points))])
    [w,centers] = rbf(points,K,y,gamma)
    [w2,centers2] = rbf(points,K2,y,gamma)
    clf = SVC(float('Inf'), 'rbf', gamma=1.5).fit(points,y)
    #plt.scatter(points[:,0],points[:,1])
    #plt.scatter(centers[:,0],centers[:,1],c='red')
    #plt.show()
    #print(w)
    #if clf.score(points,y) != 1:
    #    fail += 1
    if w == "empty" or w2 == "empty" or clf.score(points,y) != 1:
        continue
    counter += 1
    testpoints = 2 * np.random.random_sample((1000,2)) - 1
    #a = outerror(w,centers,gamma,testpoints)
```

```
    #b =
    1-clf.score(testpoints,np.array([np.sign(testpoints[i,1]-testpoints[i,0]+0.25*np.sin(np.pi*te
    stpoints[i,0])) for i in range(len(testpoints))]))
    #c = outerror(w2,centers2,gamma,testpoints)
    a = inerror(w,centers,gamma,points)
    b = outerror(w,centers,gamma,testpoints)
    c = inerror(w2,centers2,gamma,points)
    d = outerror(w2,centers2,gamma,testpoints)
    #if a == 0:
    #   reg_E_in += 1
    if c < a and d > b:
        a16 += 1
    if c > a and d < b:
        b16 += 1
    if c > a and d > b:
        c16 += 1
    if c < a and d < b:
        d16 += 1
    if c == a and d == b:
        e16 += 1
    if b < a:
        kern_better += 1
    if b < c:
        kern_better2 += 1
    #(a,b,c)
print(a16, b16, c16, d16, e16)
#print(fail/N)
print(kern_better/counter, kern_better2/counter)
```