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

runs = 1000
points = 100
d = 2
Nprob_points = 10000

def run():
    weight = np.zeros(d + 1)

    training_points = np.random.uniform(-1,1, (points, d))
    training_points = np.c_[training_points, np.zeros(points)]
    training_points = np.c_[np.ones(points), training_points]
    target_points = np.random.uniform(-1,1, (2, d))

    for point in training_points:
        det = (point[1] - target_points[0][0]) * (target_points[1][1] -
target_points[0][1]) - (point[2] - target_points[0][1]) * (target_points[1][0] -
target_points[0][0])
        if det > 0:
            # on one side
            point[3] = 1
        else:
            # on the other side
            point[3] = -1
    iterations = 0
    converged = False
    # extracts column 3, the signs
    correct_signs = training_points[:,[3]]

    training_points2 = np.copy(training_points)
    training_points2[:, 3] = 0
    test_signs = np.copy(training_points2[:,[3]])

    while (not converged):
        i = np.random.randint(points)
        if test_signs[i] != correct_signs[i]:
            weight[0] = weight[0] + training_points[i][3] * training_points[i][0]
            weight[1] = weight[1] + training_points[i][3] * training_points[i][1]
            weight[2] = weight[2] + training_points[i][3] * training_points[i][2]
            print weight
        else:
            continue
        cp1 = [0, -float(weight[0]) / weight[2]]
        cp2 = [1, float(weight[0]) * float(weight[1]) / weight[2]]

        for point in training_points2:
            det = (point[1] - cp1[0]) * (cp2[1] - cp1[1]) - (point[2] - cp1[1]) *
(cp2[0] - cp1[0])
            if det > 0:
                point[3] = 1
            else:
                point[3] = -1
        test_signs = training_points2[:,[3]]
        iterations += 1

        if np.all(np.array_equal(test_signs, correct_signs)) or iterations > 100:
```

```
            converged = True
            break
    for point in training_points:
        if point[3] == 1:
            plt.plot([point[1]], [point[2]], 'go')
        else:
            plt.plot([point[1]], [point[2]], 'ro')
        x = np.linspace(-1, 1, 100)
        plt.plot(x, (-float(weight[0]) - float(weight[1]) * x) / weight[2],'k')
        m = ((target_points[1][1] - target_points[0][1]) / (target_points[1][0] -
target_points[0][0]))
        plt.plot(x, m * x + (target_points[0][1] - m * target_points[0][0]), 'b')
        plt.axis([-1, 1, -1, 1])

        # now for probability
        prob_points = np.random.uniform(-1,1, (points, d))
        prob_points = np.c_[prob_points, np.zeros(Nprob_points)]
        inside = 0
        for point in prob_points:
            cp1 = [0, -float(weight[0]) / weight[2]]
            cp2 = [1, float(weight[0]) * float(weight[1]) / weight[2]]
            det1 = (point[1] - target_points[0][0]) * (target_points[1][1] -
target_points[0][1]) - (point[2] - target_points[0][1]) * (target_points[1][0] -
target_points[0][0])
            det2 = (point[1] - cp1[0]) * (cp2[1] - cp1[1]) - (point[2] - cp1[1]) *
(cp2[0] - cp1[0])
            if det1 > 0 and det2 <= 0:
                inside += 1
            elif det1 <= 0 and det > 0:
                inside += 1
    probability = inside / Nprob_points
    return (iterations, probability)

iterations_tot = 0
prob_tot = 0
for q in range(runs):
    iters,prob = run()
    iterations_tot += iters
    prob_tot += prob
print float(iterations_tot) / runs
print float(prob_tot) / runs
```