TH_W11_SoftMargin_SVM

Phạm Ngọc Hải - 21002139

May 6, 2024

1 Ví dụ 1.

(Dữ liệu tự tạo)
200 mẫu, 2 lớp - mỗi lớp 100 mẫu, số chiều d = 2.

1.1 Cách 1. Tự xây dựng các bước giải bài toán ràng buộc.

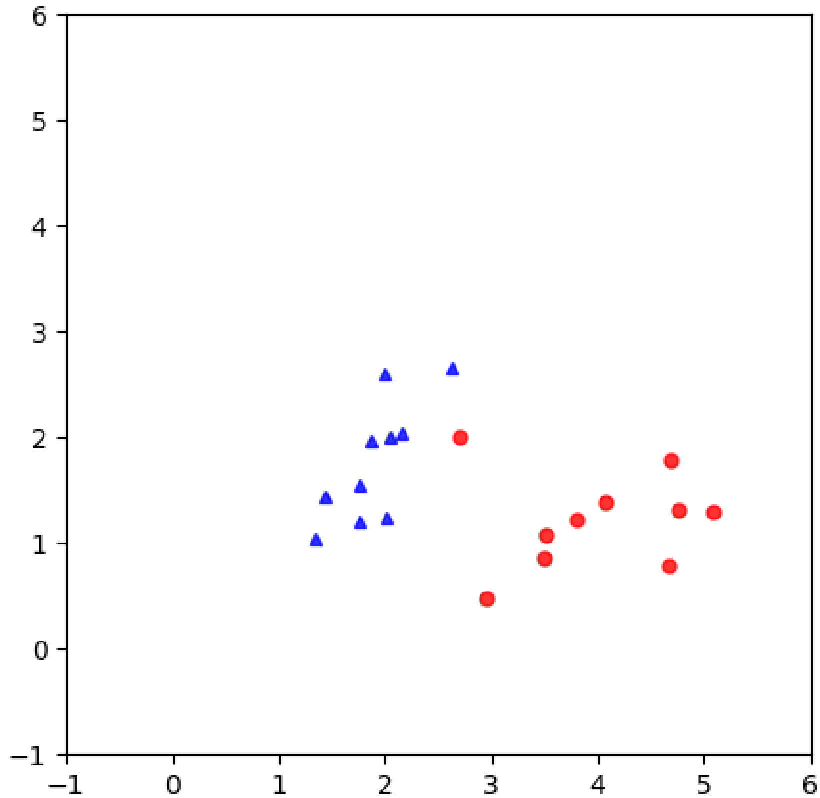
Giải bài toán Lagrange đối ngẫu $g(\lambda)$ để tìm λ sau đó tính w, w_0

Tạo data

[ ]:
```
# generate data
# list of points
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cdist
np.random.seed(21)
from matplotlib.backends.backend_pdf import PdfPages
means = [[2, 2], [4, 1]]
cov = [[.3, .2], [.2, .3]]
N = 10
X0 = np.random.multivariate_normal(means[0], cov, N)
X1 = np.random.multivariate_normal(means[1], cov, N)
X1[-1, :] = [2.7, 2]
X = np.concatenate((X0.T, X1.T), axis = 1)
y = np.concatenate((np.ones((1, N)), -1*np.ones((1, N))), axis = 1)
```

Trực quan hóa data

[ ]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(5, 5))
ani = plt.cla()
#plot points
ani = plt.plot(X0.T[0, :], X0.T[1, :], 'b^', markersize = 5, alpha = .8)
ani = plt.plot(X1.T[0, :], X1.T[1, :], 'ro', markersize = 5, alpha = .8)
ani = plt.axis([-1 , 6, -1, 6])
plt.show()
```



Giải bài toán tối tưu: $-g(\lambda) \rightarrow min$
Thực hiện tương tự như với HardMargin SVM, nhưng thêm ràng buộc chặn trên của các nhân tử Lagrange $(0 \leq \lambda \leq C)$

[ ]:
```
!pip install cvxopt
```

```
Collecting cvxopt
  Using cached
cvxopt-1.3.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(1.3 kB)
Using cached
cvxopt-1.3.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (13.6
MB)
Installing collected packages: cvxopt
Successfully installed cvxopt-1.3.2
```

[ ]:
```
from cvxopt import matrix, solvers
C = 100
# build K
V = np.concatenate((X0.T, -X1.T), axis = 1)
K = matrix(V.T.dot(V))
p = matrix(-np.ones((2*N, 1)))
```

```
# build A, b, G, h
G = matrix(np.vstack((-np.eye(2*N), np.eye(2*N))))
h = matrix(np.vstack((np.zeros((2*N, 1)), C*np.ones((2*N, 1)))))
A = matrix(y.reshape((-1, 2*N)))
b = matrix(np.zeros((1, 1)))
solvers.options['show_progress'] = False
sol = solvers.qp(K, p, G, h, A, b)
l = np.array(sol['x'])
print('lambda = \n', l.T)
```

```
lambda =
 [[1.26997770e-08 7.29907090e-09 6.75263620e+00 1.20067067e-08
  8.83482181e-09 1.00135373e-08 9.49241066e-09 1.10095260e-08
  1.09448265e-08 1.15277180e+01 3.06483278e-09 2.92217775e-09
  3.52341246e-09 5.49363383e-09 4.48478627e-09 7.55953464e-09
  2.73325320e-09 5.71296652e-09 5.02756847e-09 1.82803543e+01]]
```

Tìm tập hợp những điểm support và những điểm nằm trên margins. Lọc ra các giá trị nhỏ hơn 10^{-6} sau đó tính w, w_0

[ ]:
```
S = np.where(l > 1e-5)[0] # support set
S2 = np.where(l < .999*C)[0]
M = [val for val in S if val in S2] # intersection of two lists
XT = X.T # we need each column to be one data point in this alg
VS = V[:, S]
lS = l[S]
yM = y[M]
XM = XT[:, M]
w_dual = VS.dot(lS).reshape(-1, 1)
b_dual = np.mean(yM.T - w_dual.T.dot(XM))
print(w_dual.T, b_dual)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Cell In[7], line 7
      5 VS = V[:, S]
      6 lS = l[S]
----> 7 yM = y[M]
      8 XM = XT[:, M]
      9 w_dual = VS.dot(lS).reshape(-1, 1)

IndexError: index 2 is out of bounds for axis 0 with size 1
```

2 Bài thực hành 1

3 Bài thực hành 2

Sửa phần đọc dữ liệu trong Ví dụ 1 ở trên, sau đó áp dụng cho phần phân loại phân loại bệnh nhân ung thư vú của Đại học Wisconsin–Madison, Hoa Kỳ. Dữ liệu có sẵn trong thư viện sklearn. Dữ liệu có 569 bản ghi (mẫu), với 30 thuộc tính. Bệnh nhân được chia làm hai loại: u lành tính (B – Begnin) có 357 mẫu và u ác tính (M – Malignant) có 212 mẫu. Đoạn chương trình thực hiện việc đọc dữ liệu như dưới đây:

[ ]:
```
from sklearn import datasets

cancer_data = datasets.load_breast_cancer()
# show to test record 5th
print(cancer_data.data[5])
print(cancer_data.data.shape)
# target set
print(set(cancer_data.target))

from sklearn.model_selection import train_test_split

cancer_data = datasets.load_breast_cancer()
X_train, X_test, y_train, y_test = train_test_split(
    cancer_data.data, cancer_data.target, test_size=0.3, random_state=109
)
```

```
[1.245e+01 1.570e+01 8.257e+01 4.771e+02 1.278e-01 1.700e-01 1.578e-01
 8.089e-02 2.087e-01 7.613e-02 3.345e-01 8.902e-01 2.217e+00 2.719e+01
 7.510e-03 3.345e-02 3.672e-02 1.137e-02 2.165e-02 5.082e-03 1.547e+01
 2.375e+01 1.034e+02 7.416e+02 1.791e-01 5.249e-01 5.355e-01 1.741e-01
 3.985e-01 1.244e-01]
(569, 30)
{0, 1}
```

Thực hiện các phương pháp SVM cho dữ liệu này.

3.1 Cách 1. Tự xây dựng các bước giải bài toán ràng buộc

[ ]:
```
#!pip install cvxopt
from cvxopt import matrix, solvers
C = 100
N = len(X_train)
# build K

# Calculate V
X0 = X_train[y_train == 0]
X1 = X_train[y_train == 1]
V = np.concatenate((X0.T, -X1.T), axis=1)
```

```python
# V = X_train
K = matrix(V.T.dot(V))
p = matrix(-np.ones((N, 1)))

# build A, b, G, h
G = matrix(np.vstack((-np.eye(N), np.eye(N))))
h = matrix(np.vstack((np.zeros((N, 1)), C*np.ones((N, 1)))))
A = matrix(y_train.astype(float).reshape((-1, N)))  # y phải là float
b = matrix(np.zeros((1, 1)))
solvers.options['show_progress'] = False
sol = solvers.qp(K, p, G, h, A, b)
l = np.array(sol['x'])
print('lambda = \n', l.T)
```

```
lambda = 
 [[ 2.34837131e-16  1.54071427e-21 -1.14572257e-22  1.01377030e-21
   9.81492393e-22  5.14386652e-22  3.83818368e-22  1.41572935e-16
   1.00000000e+02  1.33112034e-21  2.65524476e-16  1.17980745e-21
  -9.12580982e-22  3.14552298e-17  1.68256094e-16  5.81062931e-17
   1.28346445e-16 -3.86127845e-22  9.14310702e-16  1.46119447e-21
   1.83514453e-16 -5.85128846e-22 -3.67279436e-22  1.33183518e-21
   1.53689591e-21  5.99737009e-16  3.04952684e-16  6.90753478e-22
   2.50325738e-17  1.75672787e-21  1.50302546e-21 -7.66609608e-22
  -6.65761891e-22  1.05384872e-21  1.98550444e-16  7.03430424e-22
   1.16648362e-15  1.79960786e-21  1.26208314e-16  7.11715442e-22
   4.13591190e-16 -6.22486112e-22  4.21491101e-16  1.15162924e-16
   1.00099665e-16  1.13043330e-16  4.79060083e-22 -2.22943209e-22
   2.63967792e-16  1.48379598e-21  6.78287471e+01  1.07893756e-21
   7.07105117e-01  1.25914146e-16  4.64608736e-22  1.79608834e+01
   1.64354821e-16  1.35940581e-21  1.91736346e-21  4.99569829e-22
   3.22377138e-22  1.42226319e-21  1.34039084e-23  2.82207854e-16
   9.34435563e+01 -9.47992315e-22  4.93743347e-21  1.33642777e-21
   1.20520697e-21  3.74338104e-22  7.39439729e-22  1.61372446e-21
   1.55874872e-21  2.22090289e-17  7.72492309e-22  1.34178799e-21
   1.11322789e-21 -3.19351393e-22  2.95518599e-16 -7.31258614e-22
   6.04947298e-22  1.07837884e-21  2.66229812e-16  4.44059414e-22
  -1.09895886e-21  1.63038689e-21  1.29165228e-21  8.01559680e-17
  -6.37709530e-22  3.74743283e-22 -6.56980344e-22  1.50959849e-21
   4.34395616e-22  9.95786642e-22  5.27106297e-22  3.85303739e-16
   1.40364308e-21 -5.29530782e-22  1.35706866e-21  9.37913143e-17
   1.72825989e-21 -2.76067723e-22  9.17743345e-17  1.27147484e-21
   6.64421790e-22 -1.31987924e-22  7.19470056e-17  1.58110312e-21
  -6.42680217e-22  4.70275623e-17  8.10564719e-17  3.35023636e-16
   1.79332424e-21  1.45872655e-21  4.01590137e-22  6.86223383e+01
  -5.51831823e-22 -4.08780751e-22  9.59764581e-22  4.55436805e-17
   6.90262216e-17  2.63925386e-22 -9.33600899e-21  3.18407161e-22
   4.24303333e-16  1.79118065e-21  7.64444621e+01  1.00000000e+02
   1.08429969e-16 -9.96158009e-22  9.33183029e-17  6.28335135e-22
```

-1.97570923e-22	1.02361667e-21	5.23713180e-17	-5.43834593e-22
2.36208179e-16	-1.72018342e-23	1.37640975e-16	4.42136247e-17
1.50826410e-21	9.97493743e-22	6.55673660e-22	1.35944042e-21
4.18423188e-17	1.54530344e-16	1.79082730e-21	1.14466952e-21
1.26439405e-21	-5.82023772e-22	-9.14031029e-22	2.23080908e-16
-9.01640750e-22	-2.99406757e-22	-1.58635039e-21	4.57859063e-15
6.71753154e-16	2.97060782e-22	2.02986941e+01	-4.10640689e-22
5.48364538e-16	3.67012582e-16	2.56578103e-16	2.81842334e-16
-3.76117053e-22	-1.29603840e-21	2.92816364e-16	7.77516742e-16
-4.25926755e-22	-4.89071325e-22	4.31617007e-22	6.58443223e-23
-8.38721706e-22	3.42425051e-22	9.63326326e-23	-6.07703412e-22
1.60568211e-22	2.93770201e-16	-6.54426119e-22	4.55917830e-16
4.16170202e-16	1.61523138e-15	-3.08668083e-22	-9.60936689e-22
3.41827249e-16	-2.35905141e-22	4.54730992e-16	7.35514657e-23
-6.81672378e-22	-1.96936065e-22	3.33435454e-16	-2.91615414e-22
-6.36966170e-22	8.94868624e-16	3.76277748e+01	3.81969290e-23
2.85905989e-16	-4.17675399e-22	-2.72396820e-22	-2.20661916e-22
6.56721689e-16	2.77623612e-16	-1.04442144e-21	4.38183827e+01
9.24330834e-23	-1.31464006e-21	-7.90681204e-23	-2.71915987e-22
3.61022811e-16	3.35081343e-22	-4.86602726e-22	-3.90188071e-22
-3.51238410e-22	4.86097767e-23	3.59054780e-16	-4.92507374e-22
-1.13301669e-21	-7.95239079e-22	8.88504457e-15	3.79764002e-16
2.77262691e-16	1.39768540e-22	4.60364985e-16	2.69658538e-16
4.62462737e-23	-5.54317737e-23	6.56398078e-23	-3.11993883e-22
1.38096296e-22	3.44493491e-16	3.36683925e-16	3.91738911e-16
-1.14765412e-22	-4.79238191e-22	-2.22721626e-22	3.79894852e-16
-1.01699167e-22	3.72788538e-16	-1.18847358e-21	-9.05773502e-23
-8.08512758e-22	-4.86842694e-22	-1.98690349e-22	2.87221938e-16
2.93965319e-16	4.21362797e-16	-7.47554768e-22	-5.56800048e-22
-4.23688168e-22	2.04768894e-15	-2.38409115e-22	4.20967627e-16
3.34922433e-16	6.61841529e-16	6.27038357e-16	2.59913316e-15
-6.33855130e-22	-4.71220058e-22	-9.65180586e-22	2.62741902e-22
6.91884414e-16	-4.14683464e-22	-1.66289919e-22	-2.54007928e-22
5.96033656e-16	-1.04202072e-21	2.20927561e-22	2.62503401e-16
-6.99382828e-22	5.23491691e-16	2.81615971e-22	2.43409407e-16
-7.94149496e-22	4.38740661e-16	-1.86840959e-23	-4.49328933e-22
9.22026228e+01	-9.06665800e-23	2.81475922e-22	-6.51646618e-22
-1.83274957e-22	-4.19135354e-22	-6.88598934e-22	2.95337873e-16
2.80026864e-16	1.00000000e+02	5.62939572e-16	-4.58347628e-22
3.19101109e-16	-1.21407961e-21	-4.00621892e-22	-7.44198315e-22
2.51744181e-16	6.34386301e-22	-1.01884815e-22	-6.60589359e-22
1.70824571e-16	-1.18158738e-22	6.33600410e-16	3.48606583e-16
5.23676733e-22	-1.11986626e-21	-9.39348121e-24	9.26772909e-16
-1.74298454e-22	2.92588558e-16	-3.45533666e-22	4.18760259e-16
2.51023164e-16	5.26872579e-16	2.90834012e-16	-3.93099101e-22
-1.78363699e-22	1.18276858e+01	6.22424011e-16	-1.05101397e-21
1.99089636e-16	2.57528048e-22	-6.33918674e-22	1.89157644e-22
3.34051785e-16	9.73382978e-23	1.47024522e+01	-4.47517884e-22

```
 -2.68102403e-22  3.93943924e-16 -4.60513344e-22 -8.38336167e-22
  3.06491855e-16  3.35877599e-15  1.02683512e-22  6.40835681e-16
 -8.75016507e-22  3.74060811e-16  1.16341049e-22  1.02878552e-15
  2.53411683e-22  2.15047909e-15  4.64536958e-16  2.02462851e-22
 -1.31711585e-21 -2.06066290e-22  2.90206659e-22  4.51253592e-23
 -3.37334043e-22 -8.01363920e-22 -5.76622575e-23  1.32571712e-22
 -2.27900612e-22 -7.53768285e-22  3.18828507e-22  6.34820872e-16
  3.43649926e-16  2.65381004e-16  1.00000000e+02 -1.15593782e-21
  3.51999820e-22  3.36936452e-22  5.10215485e-16  3.94031431e-16
  5.90643732e-22 -6.71306749e-22 -5.97217711e-22  2.07007955e-15
  2.24452118e-16 -7.76295686e-22 -2.62361030e-22 -7.85036437e-22
  9.25303797e-23  2.60989425e-16 -4.31305654e-22 -1.27145478e-21
  1.10356997e-22 -4.40883823e-22 -1.16515890e-21 -1.03285573e-22
 -7.83086941e-22 -1.04925782e-21  2.71873909e-22 -2.46033557e-22
 -1.37735777e-21  1.67505706e-16 -2.24034062e-23  4.59928004e-22
 -7.13544432e-22 -3.56363167e-22 -6.72193063e-22 -3.81317508e-22
  2.67917948e+01  2.41361472e-15  3.73508276e-16 -1.89809183e-22
 -1.04145025e-21  8.90680348e+01 -6.69371985e-22  2.18605645e-16
 -6.24849990e-22  1.17321974e-15]]
```

[ ]:
```
l.shape
```

[ ]:
```
(398, 1)
```

Lọc bỏ các lambda xấp xỉ 0

[ ]:
```
S = np.where(l > 1e-5)[0] # support set
S2 = np.where(l < .999*C)[0]
M = [val for val in S if val in S2] # intersection of two lists
VS = V[:, S]
lS = l[S]
yM = y_train[M]
XM = X_train[M, :]
w_dual = VS.dot(lS).reshape(-1, 1)
b_dual = np.mean(yM.T - w_dual.T.dot(XM.T))
print(w_dual.T, b_dual)
```

```
[[-6.57075117e-01 -4.67958059e-01  2.35297916e-01 -1.08001847e-02
   2.92323675e+00 -1.66059977e+00  2.58602340e+00  5.10950940e+00
   3.10672544e+00 -1.04737650e+00  8.84052672e+00 -1.80602444e+00
   3.38613257e-01  4.78770819e-02  1.66953983e-01 -3.18467678e+00
  -2.60220812e+00  6.90328879e-01 -7.11385151e-01 -3.84833142e-01
  -2.54906449e+00  6.24668080e-01 -1.36602407e-01  4.12118753e-02
   4.24391826e+00 -3.40335184e+00  6.51488049e+00  7.51506678e+00
   1.14306689e+01 -1.99275500e+00]] -17.991013032573417
```

[ ]:
```
print(w_dual.shape, b_dual.shape)
```

```
(30, 1) ()
```

3.2 Cách 2. Tự xây dựng các bước giải bài toán tối ưu không ràng buộc

[ ]:
```
X0.shape
```

[ ]: (149, 30)

[ ]:
```
X1.shape
```

[ ]: (249, 30)

[ ]:
```
X0_bar = np.vstack((X0.T, np.ones((1, len(X0)))))  # extended data
X1_bar = np.vstack((X1.T, np.ones((1, len(X1)))))  # extended data
Z = np.hstack((X0_bar, -X1_bar))  # as in (22)
lam = 1.0 / C


def cost(w):
    u = w.T.dot(Z)  # as in (23)
    return (np.sum(np.maximum(0, 1 - u)) + 0.5 * lam * np.sum(w * w)) - 0.5 *␣
 ↪lam * w[
        -1
    ] * w[
        -1
    ]  # no bias


def grad(w):
    u = w.T.dot(Z)  # as in (23)
    H = np.where(u < 1)[1]
    ZS = Z[:, H]
    g = -np.sum(ZS, axis=1, keepdims=True) + lam * w
    g[-1] -= lam * w[-1]  # no weight decay on bias
    return g


eps = 1e-6


def num_grad(w):
    g = np.zeros_like(w)
    for i in range(len(w)):
        wp = w.copy()
        wm = w.copy()
        wp[i] += eps
        wm[i] -= eps
        g[i] = (cost(wp) - cost(wm)) / (2 * eps)
    return g
```

```
w0 = np.random.randn(X0_bar.shape[0], 1)
g1 = grad(w0)
g2 = num_grad(w0)
diff = np.linalg.norm(g1 - g2)
print("Gradient different: %f" % diff)
```

```
Gradient different: 0.000053
```

Dùng phương pháp lặp GD và giải tìm tham số w, b

[ ]:
```
def grad_descent(w0, eta):
    w = w0
    it = 0
    while it < 100000:
        it = it + 1
        g = grad(w)
        w -= eta*g
        if (it % 10000) == 1:
            print('iter %d' %it + ' cost: %f' %cost(w))
        if np.linalg.norm(g) < 1e-5:
            break
    return w

w0 = np.random.randn(X0_bar.shape[0], 1)
w = grad_descent(w0, 0.001)
w_hinge = w[:-1].reshape(-1, 1)
b_hinge = w[-1]
print(w_hinge.T, b_hinge)
```

```
/tmp/ipykernel_7764/714090710.py:9: DeprecationWarning: Conversion of an array
with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you
extract a single element from your array before performing this operation.
(Deprecated NumPy 1.25.)
  print('iter %d' %it + ' cost: %f' %cost(w))
```

```
iter 1 cost: 47552024.006693
iter 10001 cost: 936445.736055
iter 20001 cost: 883001.301455
iter 30001 cost: 848293.353281
iter 40001 cost: 528487.466579
iter 50001 cost: 616058.394890
iter 60001 cost: 610795.016209
iter 70001 cost: 1137983.181283
iter 80001 cost: 960895.327562
iter 90001 cost: 636284.723232
[[-8.45158936e+02 -1.63970559e+02 -2.32116518e+03 -6.45811114e+01
   1.53461777e+01  8.53609352e+01  1.26657806e+02  5.27593331e+01
   2.31124834e+01  3.78736228e+00 -2.67194792e+01 -7.25198436e+01
```

```
 1.43798548e+02  4.42931335e+02  2.37716283e+00  2.29065633e+01
 3.16146571e+01  7.75840816e+00  4.56987700e+00  2.18472084e+00
-9.14330376e+02  1.43855644e+03 -4.78266904e+01  2.51863500e+02
 3.05342955e+01  2.90597817e+02  3.71684620e+02  1.04568396e+02
 7.65213954e+01  2.64020676e+01]] [-190.58458415]
```

[ ]:
```
print(w_hinge.shape, b_hinge.shape)
```

```
(30, 1) (1,)
```

3.3 Cách 3. Sử dụng thư viện

[ ]:
```
from sklearn.svm import SVC
# Copy and put code for generate data here
y = y_train.reshape((len(X_train),))
X = X_train # each sample is one row
clf = SVC(kernel = 'linear', C = 100) # use the same C# if C is small, method␣
 ↪will be "SoftMagin SVM",
# if C is large enough, method is near to hard margin
clf.fit(X, y)
w = clf.coef_
w0 = clf.intercept_
print('w = ', w)
print('b = ', w0)
```

```
w =  [[ 2.10521703e+01  2.24748145e+00 -1.77289349e+00  9.15961285e-03
  -7.50191176e+00 -4.34135787e+00 -2.25615054e+01 -1.61915144e+01
  -1.98778957e+01  5.08582561e-01 -5.57421178e+00  3.81134662e+01
  -3.92641821e+00 -1.62481829e+00 -1.18075404e+00  4.61362107e+00
  -2.22012902e+00 -3.35024278e+00 -5.99961794e+00  4.40746425e-01
   7.64789045e+00 -6.07125487e+00  2.54343023e-01 -2.28638946e-01
  -1.32196366e+01  1.01949935e+01 -4.05023898e+01 -3.32636055e+01
  -6.20170725e+01  1.96239376e+00]]
b =  [71.50072957]
```

[ ]:
```
print(w.shape, w0.shape)
```

```
(1, 30) (1,)
```

3.4 Nhận xét kết quả

Bộ trọng số của cách 1:
w = [[-6.57075117e-01 -4.67958059e-01 2.35297916e-01 -1.08001847e-02 2.92323675e+00 -1.66059977e+00 2.58602340e+00 5.10950940e+00 3.10672544e+00 -1.04737650e+00 8.84052672e+00 -1.80602444e+00 3.38613257e-01 4.78770819e-02 1.66953983e-01 -3.18467678e+00 -2.60220812e+00 6.90328879e-01 -7.11385151e-01 -3.84833142e-01 -2.54906449e+00 6.24668080e-01 -1.36602407e-01 4.12118753e-02 4.24391826e+00 -3.40335184e+00 6.51488049e+00 7.51506678e+00 1.14306689e+01 -1.99275500e+00]]
b = [-17.991013032573417]

Bộ trọng số của cách 2:
w = [[-8.45158936e+02 -1.63970559e+02 -2.32116518e+03 -6.45811114e+01 1.53461777e+01 8.53609352e+01 1.26657806e+02 5.27593331e+01 2.31124834e+01 3.78736228e+00 -2.67194792e+01 -7.25198436e+01 1.43798548e+02 4.42931335e+02 2.37716283e+00 2.29065633e+01 3.16146571e+01 7.75840816e+00 4.56987700e+00 2.18472084e+00 -9.14330376e+02 1.43855644e+03 -4.78266904e+01 2.51863500e+02 3.05342955e+01 2.90597817e+02 3.71684620e+02 1.04568396e+02 7.65213954e+01 2.64020676e+01]]
b = [-190.58458415]

Bộ trọng số của cách 3:
w = [[2.10521703e+01 2.24748145e+00 -1.77289349e+00 9.15961285e-03 -7.50191176e+00 -4.34135787e+00 -2.25615054e+01 -1.61915144e+01 -1.98778957e+01 5.08582561e-01 -5.57421178e+00 3.81134662e+01 -3.92641821e+00 -1.62481829e+00 -1.18075404e+00 4.61362107e+00 -2.22012902e+00 -3.35024278e+00 -5.99961794e+00 4.40746425e-01 7.64789045e+00 -6.07125487e+00 2.54343023e-01 -2.28638946e-01 -1.32196366e+01 1.01949935e+01 -4.05023898e+01 -3.32636055e+01 -6.20170725e+01 1.96239376e+00]]
b = [71.50072957]

Bộ trọng số cách 3 là khá tốt (chạy từ thư viện với độ chính xác cao), tuy nhiên bộ trọng số của cách 1 và 2 khác khá nhiều với cách 3 nên có thể do cách 1 và cách 2 setup chưa đủ tốt.