# **Solving the XOR problem**

Today we will consider another practical problem related to logistic regression, which is called the XOR problem.

So, unlike the previous problem, we have only four points of input data here.

import numpy as np

import matplotlib.pyplot as plt

N = 4

D = 2

XOR, which is excluding OR, has two input variables, each of them takes the value of “true” or “false.” In this case, as an output, the function produces a value of “true” if the incoming variables have different values, and “false” if the same ones.

X = np.array([

[0, 0],

[0, 1],

[1, 0],

[1, 1],

])

T = np.array([0, 1, 1, 0])

Again, we add a column of ones for our input data, and we graph it so you can see how it looks.

ones = np.array([[1]\*N]).T

plt.scatter(X[:,0], X[:,1], *c*=T)

plt.show()

You can see two blue dots and two red dots. The problem with using logistic regression is that you cannot draw a straight line that gives a satisfactory classification, since no matter what straight line you have, you will not get a classification with a coefficient more than 50%.

The solution to the XOR problem is that we create another dimension of our input data again, thus transforming the two-dimensional problem into a three-dimensional one. After that, we can easily draw a line between two data classes.

If you have some experience in building 3D images, you can easily see that by reducing the variables x and y to a new variable, we can convert the data into linearly shared ones.

xy = np.matrix(X[:,0] \* X[:,1]).T

Xb = np.array(np.concatenate((ones, xy, X), *axis*=1))

The rest of our code remains the same; we can copy it from the code of the previous programs.

w = np.random.randn(D + 2)

z = Xb.dot(w)

*def* sigmoid(*z*):

return 1/(1 + np.exp(-z))

Y = sigmoid(z)

*def* cross\_entropy(*T, Y*):

E = 0

for i in *xrange*(N)

if T[i] == 1:

E -= np.log(Y[i])

else:

E -= np.log(1 – Y[i])

return E

Let’s proceed to our code. It remains practically the same, except that the training coefficient is equal to 0.001.

learning\_rate = 0.001

error = []

for i in *xrange*(5000):

e = cross\_entropy(T, Y)

error.append(e)

if i % 100 == 0:

print e

w += learning\_rate \* ( np.dot((T – Y).T, Xb) – 0.01\*w)

Y = sigmoid(Xb.dot(w))

plt.plot(error)

plt.title(‘’Cross-entropy per iteration’’)

print ‘’Final w’’, w

print ‘’Final classification rate:’’, 1 – np.abs(T – np.round(Y)). sum / N

Let’s run the program and see what classification factor we have got.

we see the cross-entropy error function and its change with time. You might want to increase the number of iterations or change the training factor because I had to run the program several times to achieve the best classification coefficient.

Our last two examples lead to an exciting conclusion, namely, when it comes to machine learning, then, as you can see, we can apply logistic regression to a series of complex tasks by changing the parameters manually. We evaluated our data and changed the settings to improve the indicators of classification.

As for machine learning, ideally, the machine itself can learn to do such things – this is precisely what neural networks do. Therefore, in the future, I’m going to create several courses on neural networks that can automatically learn such things.