

01.112MACHINELEARNING (2017)

HW1

ZHOU XUEXUAN 1001603

Q1.

- a. Suppose a vector ϕ that is orthogonal to the hyperplane

$$\Phi = y - x_0$$

According to the given plane equation, normal vector is θ .

Then we apply vector projection formula:

$$distance = \frac{\theta \cdot v}{|\theta| \cdot |v|} |v| = \frac{y \cdot \theta - x_0 \cdot \theta}{|\theta|} = \frac{y \cdot \theta + \theta_0}{|\theta|}$$

b.
$$\begin{aligned} P(Z = k) &= \sum_{n=0}^k P(x = n) \cdot P(y = k - n) \\ &= \sum_{n=0}^k \frac{\alpha^{-n} e^{-\alpha}}{n!} \cdot \frac{\beta^{-(k-n)} e^{-\beta}}{(k-n)!} \\ &= \sum_{n=0}^k \frac{1}{n!(k-n)!} \alpha^{-n} \beta^{-(k-n)} e^{-(\alpha+\beta)} \\ &= \left(\sum_{n=0}^k C_k^n \alpha^{-n} \beta^{-(k-n)} \right) \frac{e^{-(\alpha+\beta)}}{k!} \\ &\quad \text{(According to binomial theorem)} \\ &= (\alpha + \beta)^k \frac{e^{-(\alpha+\beta)}}{k!} \end{aligned}$$

Q2.

- a. Python 3.6.1 and Theano 0.90

Q3.

a. Computed by theano and ended up with convergence

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3 import theano
4 import theano.tensor as T
5 import csv
6
7 ifile = open("linear.csv", "rb")
8 reader = csv.reader(ifile)
9 csvs = "https://www.dropbox.com/s/eqqyy9p849ewzt2/linear.csv?dl=1"
10 data = np.genfromtxt(ifile, delimiter=",")
11 X = data[:, 1:]
12 Y = data[:, 0]
13 d = X.shape[1] # columns num
14 n = X.shape[0] # data amount
15 learn_rate = 0.5 # learning rate for gradient descent
16 x = T.matrix(name="x") # feature matrix
17 y = T.vector(name="y") # response vector
18 w = theano.shared(np.zeros((d, 1)), name="w") # model parameters
19 risk = T.sum((T.dot(x, w) - y)**2) / 2 / n # empirical risk
20 grad_risk = T.grad(risk, w) # gradient of the risk
21 train_model = theano.function(inputs=[], outputs=risk, updates=[(w, w - learn_rate * grad_risk)], givens=(x:X, y:Y))
22 n_steps = 50
23 for i in range(n_steps):
24     print(train_model())
25     print(w.get_value())
26
27 ifile.close()

```

Run test2

```

0.004782864023955028
0.00478286402395503
0.00478286402395503
0.004782864023955033
[[-0.57392068]
 [ 1.35757059]
 [ 0.01527565]]

```

Process finished with exit code 0

answer return

$$y = -0.574x_1 + 1.358x_2 + 0.016x_3 - 1.883$$

b. Computed by python with numpy applying method of least squares

```

22 def compute_double_sum(x1, x2):
23     return rs
24
25 if __name__ == "__main__":
26     ifile = open("linear.csv", "rb")
27     data = np.genfromtxt(ifile, delimiter=",")
28     X1 = data[:, 1]
29     X2 = data[:, 2]
30     X3 = data[:, 3]
31     X4 = data[:, 4]
32     Y = data[:, 0]
33     n = len(X1)
34     Xtest_set = [1.47, 1.50, 1.52, 1.55, 1.57, 1.60, 1.63, 1.65, 1.68, 1.70, 1.73, 1.75, 1.78, 1.80, 1.83]
35     Ytest_set = [52.21, 53.12, 54.48, 55.84, 57.20, 58.57, 59.93, 61.29, 63.11, 64.47, 66.28, 68.10, 69.92, 72.19, 74.46]
36
37     Sum_y = sum(Y) # Sy
38     Sum_x1y = compute_double_sum(X1, Y)
39     Sum_x2y = compute_double_sum(X2, Y)
40     Sum_x3y = compute_double_sum(X3, Y)
41     Sum_x1 = sum(X1) # Sx1
42     Sum_x2 = sum(X2) # Sx2
43     Sum_x3 = sum(X3) # Sx3
44     Sum_x1sq = compute_double_sum(X1, X1)
45     Sum_x2sq = compute_double_sum(X2, X2)
46     Sum_x3sq = compute_double_sum(X3, X3)
47     Sum_x1x2 = compute_double_sum(X1, X2)
48     Sum_x1x3 = compute_double_sum(X1, X3)
49     Sum_x2x3 = compute_double_sum(X2, X3)
50     a = np.array([n, Sum_x1, Sum_x2, Sum_x3], [Sum_x1, Sum_x1sq, Sum_x1x2, Sum_x1x3], [Sum_x2, Sum_x1x2, Sum_x2sq, Sum_x2x3], [Sum_x3, Sum_x1x3, Sum_x2x3, Sum_x3sq])
51     matrix1 = inv(a)
52     matrix2 = np.array([Sum_y, Sum_x1y, Sum_x2y, Sum_x3y])
53     ans = np.dot(matrix1, matrix2)
54     print(ans)

```

Run test2

```

/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6 /Users/zhouxuexuan/PycharmProjects/LR/test2.py
[[-1.88288076]
 [-0.57392068]
 [ 1.35757059]
 [ 0.01527565]]

```

Process finished with exit code 0

answer return

$$y = -1.883 - 0.574x_1 + 1.358x_2 + 0.015x_3$$

c. Computed by python with statsmodels.api

```

import numpy as np
import statsmodels.api as sm

def linear_regression(x, y):
    ones = np.ones(len(x))
    X = sm.add_constant(np.column_stack((x, ones)))
    for ele in x[1:]:
        X = sm.add_constant(np.column_stack((ele, X)))
    results = sm.OLS(y, X).fit()
    return results

if __name__ == "__main__":
    ifile = open("linear.csv", "rb")
    data = np.genfromtxt(ifile, delimiter=",")
    X1 = data[:, 1]
    X2 = data[:, 2]
    X3 = data[:, 3]
    X4 = data[:, 4]
    Y = data[:, 0]
    n = len(X1)
    x_matrix = [X3, X2, X1]
    print(sm.OLS(Y, x_matrix).summary())

```

Run test3

	coef	std err	t	P> t	[0.025	0.975]
x1	-0.5739	0.015	-38.744	0.000	-0.604	-0.544
x2	1.3576	0.015	93.120	0.000	1.328	1.387
x3	0.0153	0.016	0.948	0.348	-0.017	0.048
const	-1.8829	0.015	-127.263	0.000	-1.913	-1.853

Omnibus: 1.092 Durbin-Watson: 2.451
 Prob(Omnibus): 0.579 Jarque-Bera (JB): 0.690
 Skew: 0.286 Prob(JB): 0.708
 Kurtosis: 3.058 Cond. No. 1.31

Warnings:
 [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Process finished with exit code 0

answer return

$$y = -0.574x_1 + 1.358x_2 + 0.015x_3 - 1.8829$$

Conclusion:

All three methods return almost exactly the same answers as showing above which is suggest that the theano training method has fairly well-performed accuracy.