

목차

1. $X=[\text{homework}, \text{attendance}, \text{final}]$, $y=\text{score}$ 일 때, linear regression 을 least square method(statsmodel 사용)을 사용하여 구현하고, 다양한 통계적 검증 결과를 설명하라.
2. 위의 linear regression 을 gradient descent 알고리즘을 구현하여 실행하고, 그 결과 값을 least square 방법과 비교하여라.

* 모든 목차는 실행 결과 캡처 화면 이미지를 포함

* db_conn.py: 사전에 작성해둔 database 연결 소스코드

-
1. $X=[\text{homework}, \text{attendance}, \text{final}]$, $y=\text{score}$ 일 때, linear regression 을 least square method(statsmodel 사용)을 사용하여 구현하고, 다양한 통계적 검증 결과를 설명하라.

- 소스 코드

```
import numpy as np
from db_conn import *
import statsmodels.api as sm
import matplotlib.pyplot as plt

def load_db_score_data():
    conn, cur = open_db()

    sql = "select * from score"
    cur.execute(sql)

    data = cur.fetchall()

    close_db(conn, cur)

    X = [(t['homework'], t['attendance'], t['final']) for t in data]
    X = np.array(X)
    X = sm.add_constant(X)

    y = [(t['score']) for t in data]
    y = np.array(y)
```

```

    return X, y

def least_square(X, y):
    # Least Square 메소드
    model = sm.OLS(y, X)
    results = model.fit() # fitting: data를 가지고 학습
    print(results.summary())

if __name__ == '__main__':
    X, y = load_db_score_data()
    least_square(X, y)
    print("least square _ end")

```

- 실행 결과

```

Dep. Variable: y R-squared: 0.825
Model: OLS Adj. R-squared: 0.819
Method: Least Squares F-statistic: 137.9
Date: Thu, 10 Nov 2022 Prob (F-statistic): 3.76e-33
Time: 02:06:11 Log-Likelihood: -300.38
No. Observations: 92 AIC: 608.8
Df Residuals: 88 BIC: 618.8
Df Model: 3
Covariance Type: nonrobust
=====
               coef    std err          t      P>|t|      [0.025    0.975]
-----
const      -3.6583    15.619     -0.234     0.815    -34.697    27.380
x1           1.7178     0.280      6.135     0.000      1.161     2.274
x2           1.9348     2.236      0.865     0.389     -2.508     6.377
x3           1.4972     0.091     16.481     0.000      1.317     1.678
=====
Omnibus:            0.200   Durbin-Watson:           1.442
Prob(Omnibus):      0.905   Jarque-Bera (JB):         0.296
Skew:               -0.105   Prob(JB):                 0.862
Kurtosis:           2.818   Cond. No.                  584.
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
least square _ end

```

results.summary() 출력 결과에 따라, 우선, $\text{score} = 1.7178 \cdot \text{homework} + 1.9348 \cdot \text{attendance} + 1.4972 \cdot \text{final} - 3.6583$ 이 된다. Df Residuals 은 degree of freedom 을 의미하며, 88 은 총 데이터 수인 92 에서 X variable 3 개와 constant 1 개를 더한 4 를 뺀 값이다. R-squared 는 결정 계수로, 0.825 라는 꽤 높은 수치가 결과로 나왔다.

F-statistic 은 식의 유의미한 정도를 나타낸다. 이는 2 개의 distribution 을 비교할 때, F-statistic value 가 x 좌표 상의 포인트이며, 그에 대한 확률 값을 나타내는 것이다. Prob (F-statistic)는

'데이터 분포와 만들어 놓은 linear regression 이 전혀 관련이 없다.'는 귀무가설을 기각하지 않을 확률로, 그 값이 작을수록 귀무가설을 강력하게 기각한다. 귀무가설을 기각한다는 것은, 유의미한 linear regression 임을 의미하며, $3.76e-33 (= 3.76 * 10^{-33})$ 의 수치로 매우 작은 값이 나타났기에 이는 유의미한 linear regression 을 뜻한다.

std err 값은 작을수록 variable 에 대한 예측치가 높다는 것이다. 위 결과에서는 $x_3(0.091)$, $x_1(0.280)$, $x_2(2.236)$, $const(15.619)$ 순으로 유의미하다는 것을 알 수 있다.

이와 비슷하게, t-distribution 상에서 t 값이 클수록 귀무가설 기각 확률 높아진다. 즉, t 값이 클수록 linear regression 이 유의미하다는 것을 의미한다. t 값에 따른 p-value 은 95% 신뢰도 구간에 대해서는 0.05 가 기준이다. 그 값이 0.05 보다 작으면 귀무가설을 기각하는데, 0 이라는 것은 그만큼 강하게 귀무가설 기각하는 것으로, coefficient 값을 무조건 받아들인다는 것을 의미한다. 따라서, x_1 과 x_3 은 유의미한 variable 이 되며, x_2 는 0.389, const 는 0.815 로 0.05 보다 큰 값이므로, 귀무가설 채택한다. 따라서, 이는 0 으로 보는 것과 크게 다르지 않다.(유의미하지 않다.)

위에서 서술한 바들에 의하여, x_1 에 해당하는 homework 와 x_3 에 해당하는 homework 가 굉장히 유의미하며, x_2 에 해당하는 attendance 는 유의미하지 않다는 것을 알 수 있다. 이는, homework, attendance, final 3 개를 가지고 linear regression 설계한 것이 homework 과 final, 2 개만 가지고 linear regression 설계한 것과 별반 다를 것이 없다는 것을 의미한다.

+) 95% 신뢰 구간에 대하여 [0.025 0.975] 결과값

- ➔ const 값은 -34.697~27.380 사이에서 폭 넓게 존재
- ➔ x_3 의 경우 1.317~1.678 로 범위가 굉장히 좁게 존재 = 정확하게 예측하고 있음

2. 위의 linear regression 을 gradient descent 알고리즘을 구현하여 실행하고, 그 결과 값을 least square 방법과 비교하여라.

- 소스 코드

```
import numpy as np
from db_conn import *
import statsmodels.api as sm
import matplotlib.pyplot as plt

def load_db_score_data():
    conn, cur = open_db()

    sql = "select * from score"
    cur.execute(sql)

    data = cur.fetchall()

    close_db(conn, cur)

    X = [(t['homework'], t['attendance'], t['final']) for t in data]
    X = np.array(X)

    y = [(t['score']) for t in data]
    y = np.array(y)

    return X, y

def least_square(X, y):
    X_const = sm.add_constant(X)

    model = sm.OLS(y, X_const)
    ls = model.fit() # ls = least square

    ls_c = ls.params[0]
    ls_m1 = ls.params[1]
    ls_m2 = ls.params[2]
    ls_m3 = ls.params[3]

    return ls_m1, ls_m2, ls_m3, ls_c

def gradient_descent(X, y):
    epochs = 200000
    min_grad = 0.00001
```

```

learning_rate = 0.001

m1 = 0.0
m2 = 0.0
m3 = 0.0
c = 0.0
n = len(y) # data 의 개수

for epoch in range(epochs):
    m1_partial = 0.0 # m 에 대한 편미분 값
    m2_partial = 0.0 # m 에 대한 편미분 값
    m3_partial = 0.0 # m 에 대한 편미분 값
    c_partial = 0.0 # c 에 대한 편미분 값

    for i in range(n):
        y_pred = m1 * X[i][0] + m2 * X[i][1] + m3 * X[i][2] + c
        m1_partial += (y_pred - y[i])*X[i][0]
        m2_partial += (y_pred - y[i])*X[i][1]
        m3_partial += (y_pred - y[i])*X[i][2]
        c_partial += (y_pred - y[i])

    m1_partial *= 2/n
    m2_partial *= 2/n
    m3_partial *= 2/n
    c_partial *= 2/n

    delta_m1 = -learning_rate * m1_partial
    delta_m2 = -learning_rate * m2_partial
    delta_m3 = -learning_rate * m3_partial
    delta_c = -learning_rate * c_partial

    if abs(delta_m1) < min_grad and abs(delta_m2) < min_grad and
abs(delta_m3) < min_grad and abs(delta_c) < min_grad:
        print("some value is under min_grad")
        break

    m1 += delta_m1
    m2 += delta_m2
    m3 += delta_m3
    c += delta_c

    if epoch % 5000 == 0:

```

```

        print("epoch %d: delta_m1= %f, delta_m2= %f, delta_m3= %f,
delta_c= %f, m1= %f, m2= %f, m3= %f, c= %f" %(epoch, delta_m1, delta_m2,
delta_m3, delta_c, m1, m2, m3, c))

    return m1, m2, m3, c

if __name__ == '__main__':
    X, y = load_db_score_data()

    print("least square:")
    ls_m1, ls_m2, ls_m3, ls_c = least_square(X, y)
    print("ls_m1 = %f, ls_m2 = %f, ls_m3 = %f, ls_c = %f" %(ls_m1, ls_m2,
ls_m3, ls_c))

    print()
    print("gradient descent:")
    gd_m1, gd_m2, gd_m3, gd_c = gradient_descent(X, y)
    print()
    print("gd_m1 = %f, gd_m2 = %f, gd_m3 = %f, gd_c = %f" %(gd_m1, gd_m2,
gd_m3, gd_c))

```

- 실행 결과

```

linear_regression.py 1 | gradient_descent.py 1 X | db_conn.py
과제 > 4. Linear Regression > gradient_descent.py
84
85
86 if __name__ == '__main__':
87     X, y = load_db_score_data()
88
89     print("least square:")
90     ls_m1, ls_m2, ls_m3, ls_c = least_square(X, y)
91     print("ls_m1 = %f, ls_m2 = %f, ls_m3 = %f, ls_c = %f" %(ls_m1, ls_m2, ls_m3, ls_c))
92
93     print()
94     print("gradient descent:")
95     gd_m1, gd_m2, gd_m3, gd_c = gradient_descent(X, y)
96     print()
97     print("gd_m1 = %f, gd_m2 = %f, gd_m3 = %f, gd_c = %f" %(gd_m1, gd_m2, gd_m3, gd_c))

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```

least square:
ls_m1 = 1.717757, ls_m2 = 1.934755, ls_m3 = 1.497205, ls_c = -3.658348

gradient descent:
epoch 0: delta_m1= 2.385090, delta_m2= 1.003714, delta_m3= 2.010578, delta_c= 0.128088, m1= 2.385090, m2= 1.003714, m3= 2.010578, c= 0.128088
epoch 5000: delta_m1= -0.000000, delta_m2= 0.000002, delta_m3= -0.000000, delta_c= -0.000014, m1= 1.730210, m2= 1.416802, m3= 1.500513, c= 0.116160
epoch 10000: delta_m1= -0.000000, delta_m2= 0.000002, delta_m3= -0.000000, delta_c= -0.000014, m1= 1.729965, m2= 1.426264, m3= 1.500452, c= 0.047541
epoch 15000: delta_m1= -0.000000, delta_m2= 0.000002, delta_m3= -0.000000, delta_c= -0.000013, m1= 1.729743, m2= 1.435509, m3= 1.500393, c= -0.019837
epoch 20000: delta_m1= -0.000000, delta_m2= 0.000002, delta_m3= -0.000000, delta_c= -0.000013, m1= 1.729525, m2= 1.444586, m3= 1.500335, c= -0.085990
epoch 25000: delta_m1= -0.000000, delta_m2= 0.000002, delta_m3= -0.000000, delta_c= -0.000013, m1= 1.729311, m2= 1.453498, m3= 1.500278, c= -0.150940
epoch 30000: delta_m1= -0.000000, delta_m2= 0.000002, delta_m3= -0.000000, delta_c= -0.000013, m1= 1.729101, m2= 1.462248, m3= 1.500222, c= -0.214709
epoch 35000: delta_m1= -0.000000, delta_m2= 0.000002, delta_m3= -0.000000, delta_c= -0.000012, m1= 1.728894, m2= 1.470839, m3= 1.500168, c= -0.277319
epoch 40000: delta_m1= -0.000000, delta_m2= 0.000002, delta_m3= -0.000000, delta_c= -0.000012, m1= 1.728692, m2= 1.479273, m3= 1.500114, c= -0.338790
epoch 45000: delta_m1= -0.000000, delta_m2= 0.000002, delta_m3= -0.000000, delta_c= -0.000012, m1= 1.728493, m2= 1.487554, m3= 1.500061, c= -0.399144
epoch 50000: delta_m1= -0.000000, delta_m2= 0.000002, delta_m3= -0.000000, delta_c= -0.000012, m1= 1.728298, m2= 1.495685, m3= 1.500009, c= -0.458400
epoch 55000: delta_m1= -0.000000, delta_m2= 0.000002, delta_m3= -0.000000, delta_c= -0.000012, m1= 1.728106, m2= 1.503668, m3= 1.499958, c= -0.516579
epoch 60000: delta_m1= -0.000000, delta_m2= 0.000002, delta_m3= -0.000000, delta_c= -0.000011, m1= 1.727918, m2= 1.511506, m3= 1.499908, c= -0.573701
epoch 65000: delta_m1= -0.000000, delta_m2= 0.000002, delta_m3= -0.000000, delta_c= -0.000011, m1= 1.727733, m2= 1.519201, m3= 1.499859, c= -0.629783
epoch 70000: delta_m1= -0.000000, delta_m2= 0.000001, delta_m3= -0.000000, delta_c= -0.000011, m1= 1.727552, m2= 1.526756, m3= 1.499811, c= -0.684847
epoch 75000: delta_m1= -0.000000, delta_m2= 0.000001, delta_m3= -0.000000, delta_c= -0.000011, m1= 1.727374, m2= 1.534174, m3= 1.499763, c= -0.738909
epoch 80000: delta_m1= -0.000000, delta_m2= 0.000001, delta_m3= -0.000000, delta_c= -0.000011, m1= 1.727199, m2= 1.541457, m3= 1.499717, c= -0.791988
epoch 85000: delta_m1= -0.000000, delta_m2= 0.000001, delta_m3= -0.000000, delta_c= -0.000010, m1= 1.727027, m2= 1.548608, m3= 1.499671, c= -0.844102
epoch 90000: delta_m1= -0.000000, delta_m2= 0.000001, delta_m3= -0.000000, delta_c= -0.000010, m1= 1.726859, m2= 1.555628, m3= 1.499626, c= -0.895268
some value is under min_grad

gd_m1 = 1.726733, gd_m2 = 1.560852, gd_m3 = 1.499593, gd_c = -0.933342

```

master Python 3.10.8 64-bit 0 2 Ln 87, Col 31 Spaces: 4 UTF-8 LF Python Go Live Prettier

위 결과에 따르면, least square method 를 이용한 결과와 gradient descent 알고리즘의 결과 중, m_1 값과 m_3 값이 매우 유사하지만, m_2 와 constant 값은 비교적 크게 차이나는 것을 알 수 있다.