데이터사이언스 _ Linear Regression

201911278 정경은

목차

1. X=[homework, attendance, final], y=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

---

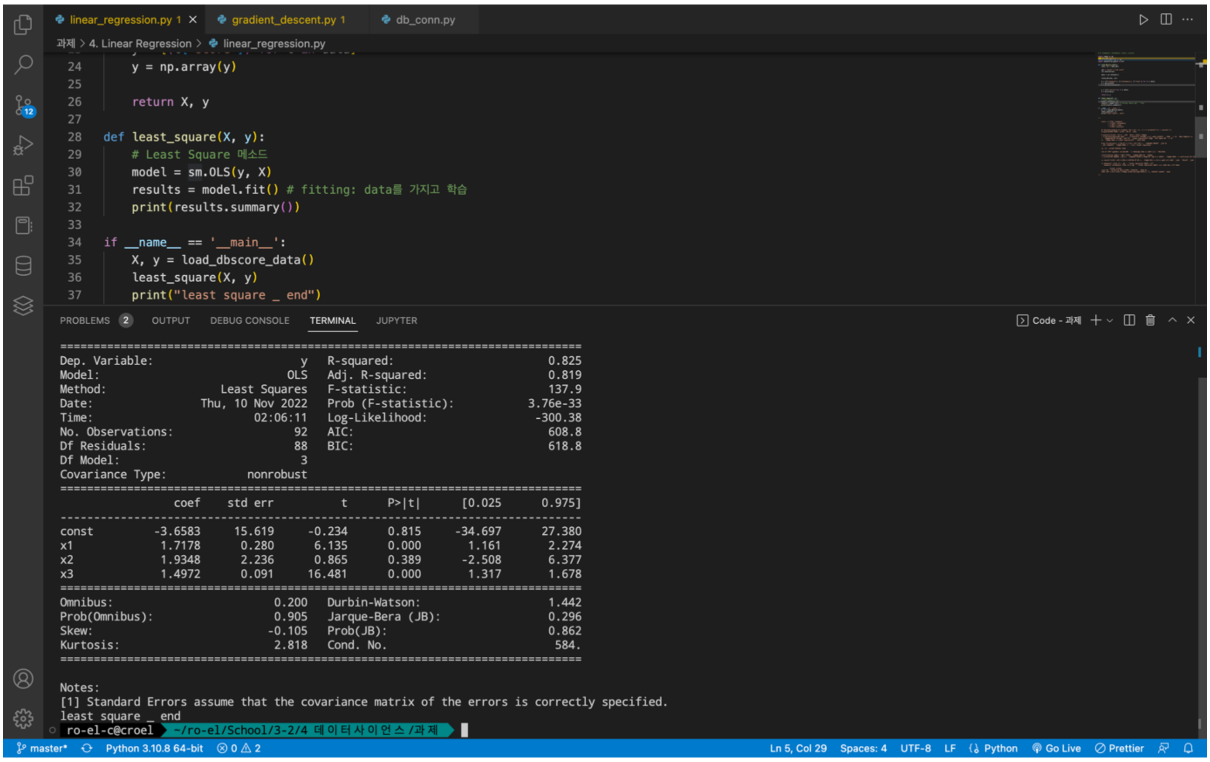
1. X=[homework, attendance, final], y=score 일 때, linear regression 을 least square method(statsmodel 사용)을 사용하여 구현하고, 다양한 통계적 검증 결과를 설명하라.

- 소스 코드

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

def load_dbscore_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)
```

```python
    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_dbscore_data()
    least_square(X, y)
    print("least square _ end")
```

- 실행 결과



results.summary() 출력 결과에 따라, 우선, score = 1.7178*homework + 1.9348*attendance + 1.4972*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 x3(0.091), x1(0.280), x2(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, x1 과 x3 은 유의미한 variable 이 되며, x2 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, x1 에 해당하는 homework 와 x3 에 해당하는 homework 가 굉장히 유의미하며, x2 에 해당하는 attendance 는 유의미하지 않다는 것을 알 수 있다. 이는, homework, attendence, 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
- ➔ x3 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.

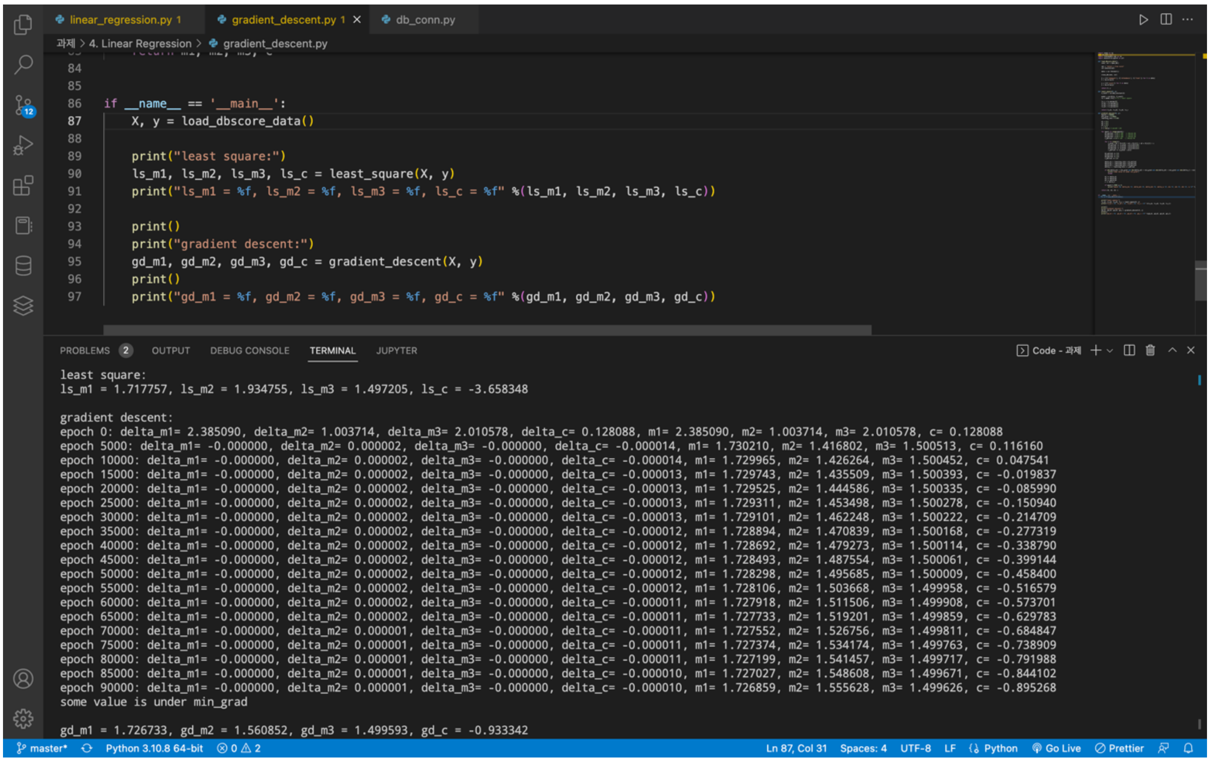
- 소스 코드

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

def load_dbscore_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
```

```python
    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:
```

```python
            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_dbscore_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



위 결과에 따르면, least square method 를 이용한 결과와 gradient descent 알고리즘의 결과 중, m1 값과 m3 값이 매우 유사하지만, m2 와 constant 값은 비교적 크게 차이나는 것을 알 수 있다.