개별연구 숙제 수행 결과 보고

20170604 정지형

GOAL of Assignment

Comparing(?) OLS/GD/MBGD/SGD
With Linear Regression

계산 시간과 정확도 비교를 비교하여 각 Method 비교

Codes

- Imports & settings

```python
# imports
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt
import time
```

- Numpy 이용
- Time – 계산 시간 측정

```python
# Generate random data-set
np.random.seed(0)
print("<Generating Data....>")
n = 100                                             # vector size
N = (2**5)*5*100                                    # number of data

x = np.random.rand(N, n)                            # Data matrix
w = 10 * np.random.rand(n, 1)                       # True parameter
yt = np.matmul(x, w)                                # y value matrix induced by data x
e = np.random.randn(N, 1)                           # error term
yd = np.add(yt, e)                                  # y data matrix we have

a = 10                                              # number of calculations
```

Codes

- Calculating solutions (Ex. OLS)

```
# Calculate the Ordinary Least Squares solution
m_time_OLS = np.zeros(a)
m_R2score_OLS = np.zeros(a)

print("<Executing OLS method....>")
for i in range(a):

    calstarted_OLS = time.time()

    b1 = inv(np.matmul(x.T, x))
    b2 = np.matmul(x.T, yd)
    B = np.matmul(b1, b2)                          # solution matrix
    y_OLS = np.matmul(x, B)                        # y value induced by x, B

    calended_OLS = time.time()
    caltime_OLS = calended_OLS - calstarted_OLS    # OLS calculation time

    m_time_OLS[i] = caltime_OLS                    # Recording caltime to get average

# Calculating R^2 score for OLS
    ssr = np.sum((yt - y_OLS)**2)
    sst = np.sum((yt - np.mean(yt))**2)
    r2score_OLS = 1 - (ssr/sst)

    m_R2score_OLS[i] = r2score_OLS                 # Recording R^2 score to get average

# Calculating average run-time of OLS
avg_time_OLS = np.average(m_time_OLS)
avg_R2score_OLS = np.average(m_R2score_OLS)
```

- 여러 번 돌려 평균 값으로 비교
- 계산 시간 측정 > 어느 method가 더 빠를까?
- R^2 Score을 통한 Evaluation

Spoiler?

- Shocking calculation results

```
C:\Users\nokn1\AppData\Local\Programs\Python\Python37-32\p
<Generating Data....>
<Executing OLS method....>
<Executing Gradient Descent....>
<Executing Mini-Batch Gradient Descent....>
<Executing Stochastic Gradient Descent....>
<Report>
| Method || Average Run-time || Average R^2 Score |
|  OLS   ||0.017554 sec      ||0.999978           |
|  GD    ||0.071709 sec      ||0.999915           |
|  MBGD  ||0.701330 sec      ||0.999685           |
|  SGD   ||0.652762 sec      ||0.919777           |
```

With n = 100, N = 16,000, a = 10

OLS → GD: 4.2 times

GD → MBGD: 9.9 times

MBGD → SGD: Similar

Spoiler?

- Shocking calculation results

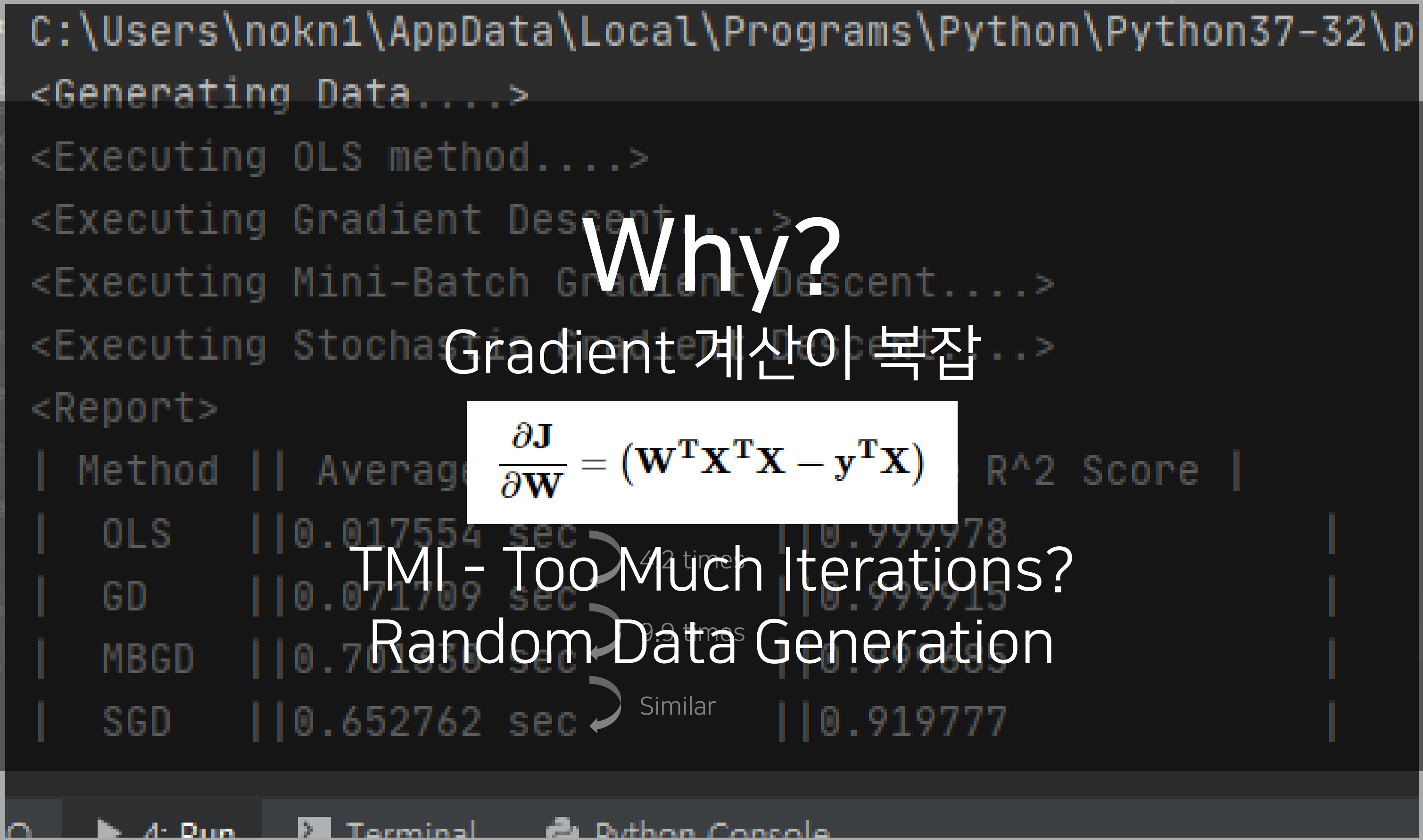
```
C:\Users\nokn1\AppData\Local\Programs\Python\Python37-32\python.exe "C:/Users/nokn1/Desktop/
<Generating Data....>
<Executing OLS method....>
<Executing Gradient Descent....>
<Report>  With n = 100, N = 1,440,000, a = 100
| Method || Average Run-time || Average R^2 Score |
|  OLS   ||2.066714 sec       ||1.000000           |
|  GD    ||1.696927 sec       ||0.999946           |

Process finished with exit code 0
```

* Another execution of methods gave a result that avg runtime GD > avg runtime OLS, so we need to check this in environment that our data has much higher dimension

Spoiler?

- Shocking calculation results



Why?

Gradient 계산이 복잡

$$\frac{\partial \mathbf{J}}{\partial \mathbf{W}} = (\mathbf{W^T X^T X} - \mathbf{y^T X})$$

TMI - Too Much Iterations?

Random Data Generation

Codes

- OLS again

```python
# Calculate the Ordinary Least Squares solution
m_time_OLS = np.zeros(a)
m_R2score_OLS = np.zeros(a)

print("<Executing OLS method....>")
for i in range(a):

    calstarted_OLS = time.time()

    b1 = inv(np.matmul(x.T, x))
    b2 = np.matmul(x.T, yd)
    B = np.matmul(b1, b2)                          # solution matrix
    y_OLS = np.matmul(x, B)                        # y value induced by x, B

    calended_OLS = time.time()
    caltime_OLS = calended_OLS - calstarted_OLS    # OLS calculation time

    m_time_OLS[i] = caltime_OLS                    # Recording caltime to get average

# Calculating R^2 score for OLS
    ssr = np.sum((yt - y_OLS)**2)
    sst = np.sum((yt - np.mean(yt))**2)
    r2score_OLS = 1 - (ssr/sst)

    m_R2score_OLS[i] = r2score_OLS                 # Recording R^2 score to get average

# Calculating average run-time of OLS
avg_time_OLS = np.average(m_time_OLS)
avg_R2score_OLS = np.average(m_R2score_OLS)
```

- Inverse Calculation in numpy Pseudo inverse를 계산
- 당연히 기존 inverse 계산보다 빠름
- So, 계산이 matrix 몇 번 곱하기에 그침

Codes

- But Gradient Descent (BGD?)

```
class LinearRegressionUsingGD:

    def __init__(self, eta=0.05, epochs=1000):
        self.eta = eta
        self.epochs = epochs

    def fit(self, x, y):

        self.w_ = np.zeros((x.shape[1], 1))
        m = x.shape[0]
        xTx = np.matmul(x.T, x)
        yTx = np.matmul(y.T, x)

        for _ in range(self.epochs):
            gradient_vector = np.matmul(self.w_.T, xTx) - yTx
            self.w_ -= (self.eta / m) * gradient_vector.T

        return self

    def predict(self, x):

        return np.dot(x, self.w_)

class LinearRegressionUsingMBGD:
```

- Learning Rate eta = 0.05
- 1,000 Epochs(iterations)
- Initial W0 = 0

- 주어진 Gradient 계산 식으로 Gradient 계산
- OLS 보다 훨씬 많은 matrix 곱셈 수행
- **If Data size is Large, OLS/GD is Comparable.**

Codes

- Mini-batch Gradient Descent

```python
class LinearRegressionUsingMBGD:

    def __init__(self, eta=0.05, epochs=1000, batch_size=32):
        self.eta = eta
        self.epochs = epochs
        self.batch_size = batch_size

    def fit(self, x, y):
        self.w_ = np.zeros((x.shape[1], 1))
        m = self.batch_size

        for _ in range(self.epochs):

            # Choosing Mini-Batch with size m
            numdata = np.arange(x.shape[0])
            J = np.random.choice(numdata, size=m, replace=False)
            xb = x[J]
            yb = y[J]
            xbTxb = np.matmul(xb.T, xb)
            ybTxb = np.matmul(yb.T, xb)
            # Calculating & Updating Gradient
            gradient_vector = np.matmul(self.w_.T, xbTxb) - ybTxb
            self.w_ -= (self.eta / m) * gradient_vector.T

        return self

    def predict(self, x):

        return np.dot(x, self.w_)
```

- Learning Rate eta = 0.05
- 1,000 Epochs(iterations)
- Batch size = 32
- Initial W0 = 0

- 1개의 Mini Batch Random 추출
- 주어진 Gradient 계산 식으로 Gradient 계산(하기 위해…)
- OLS/GD 보다 훨씬 더더 많은 matrix 곱셈 수행

Codes

- Stochastic Gradient Descent

```python
class LinearRegressionUsingSGD:

    def __init__(self, eta=0.05, epochs=1000, batch_size=1):
        self.eta = eta
        self.epochs = epochs
        self.batch_size = batch_size

    def fit(self, x, y):
        self.w_ = np.zeros((x.shape[1], 1))
        m = self.batch_size

        for _ in range(self.epochs):

            # Choosing a data randomly
            numdata = np.arange(x.shape[0])
            J = np.random.choice(numdata, size=m, replace=False)
            xb = x[J]
            yb = y[J]
            xbTxb = np.matmul(xb.T, xb)
            ybTxb = np.matmul(yb.T, xb)
            # Calculating & Updating Gradient
            gradient_vector = np.matmul(self.w_.T, xbTxb) - ybTxb
            self.w_ -= (self.eta / m) * gradient_vector.T

        return self

    def predict(self, x):

        return np.dot(x, self.w_)
```

- Learning Rate eta = 0.05
- 1,000 Epochs(iterations)
- Batch size = 1
- Initial W0 = 0

- 1개의 Data Random 추출
- 주어진 Gradient 계산식으로 Gradient 계산(하기 위해…)
- OLS/GD 보다 훨씬 더더 많은 matrix 곱셈 수행, MBGD와는 비슷.

Any approaches?

- Focusing on “Why?”

1. Gradient Calculation Formula
   - **Main Problem** of the code
   - 하지만 해결 방법을 아직 고안하지 못함

Any approaches?

- Focusing on “Why?”

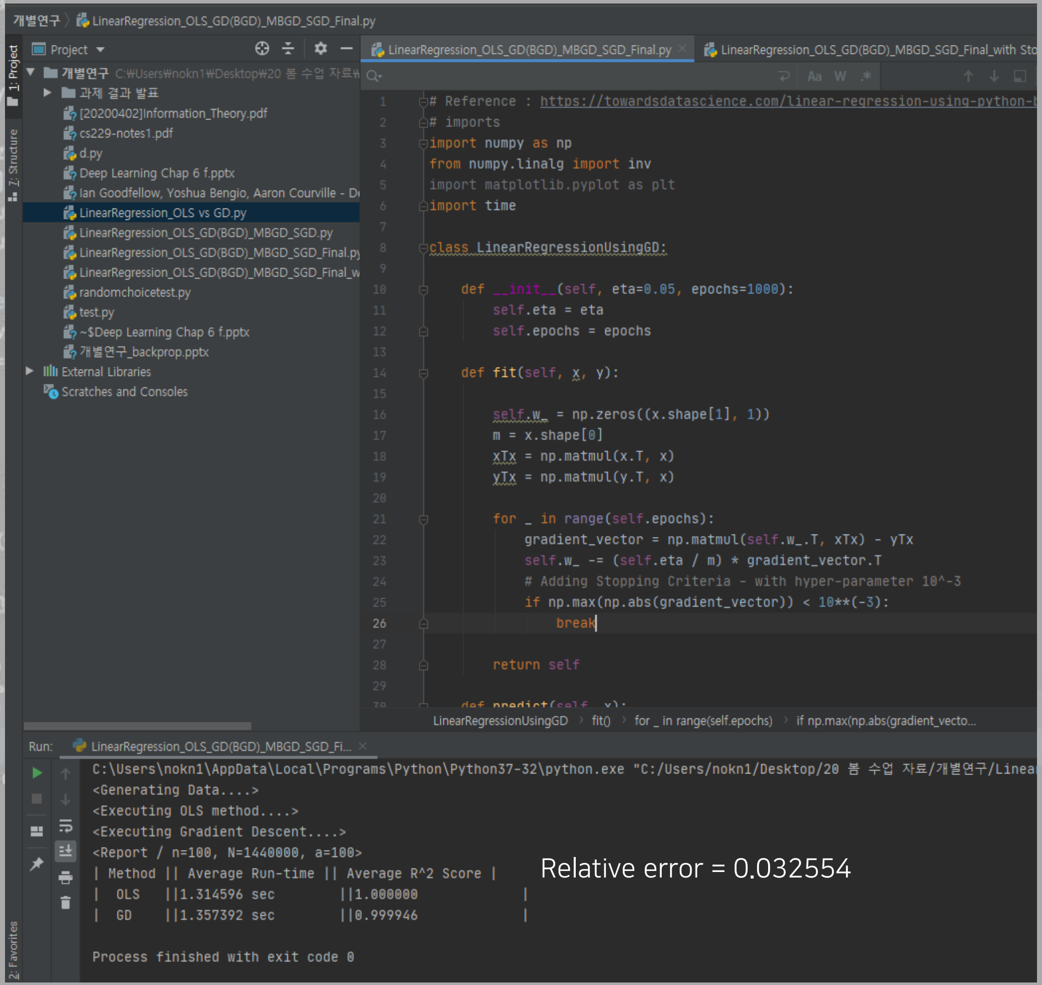
2. TMI – Too Much Iterations

1,000 epochs(iterations) > Isn’t it too large?

Stopping Criteria – gradient가 적당히 작아졌을 때 멈추도록 하면 어떨까?

Stopping Criteria

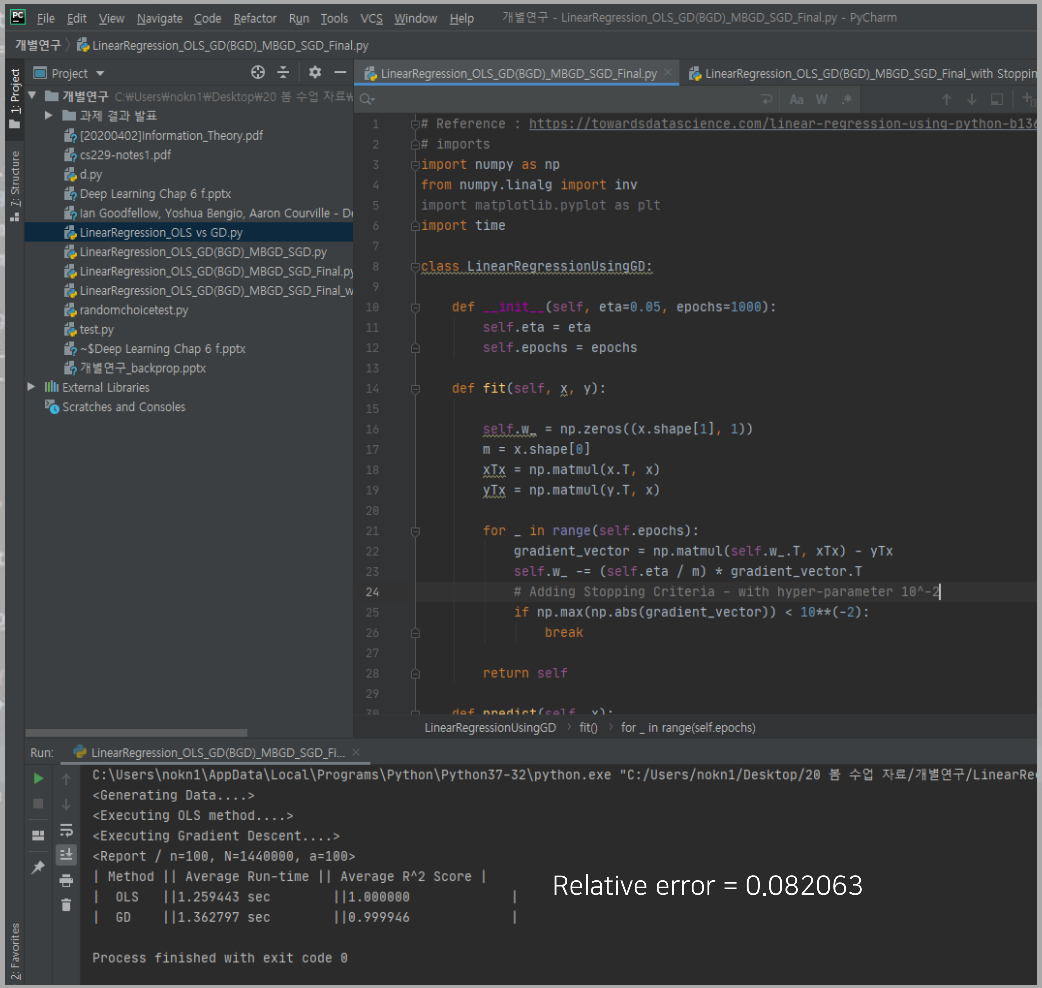
- Handling “TMI” Problem on GD case



Relative error = 0.032554

Stopping Criteria

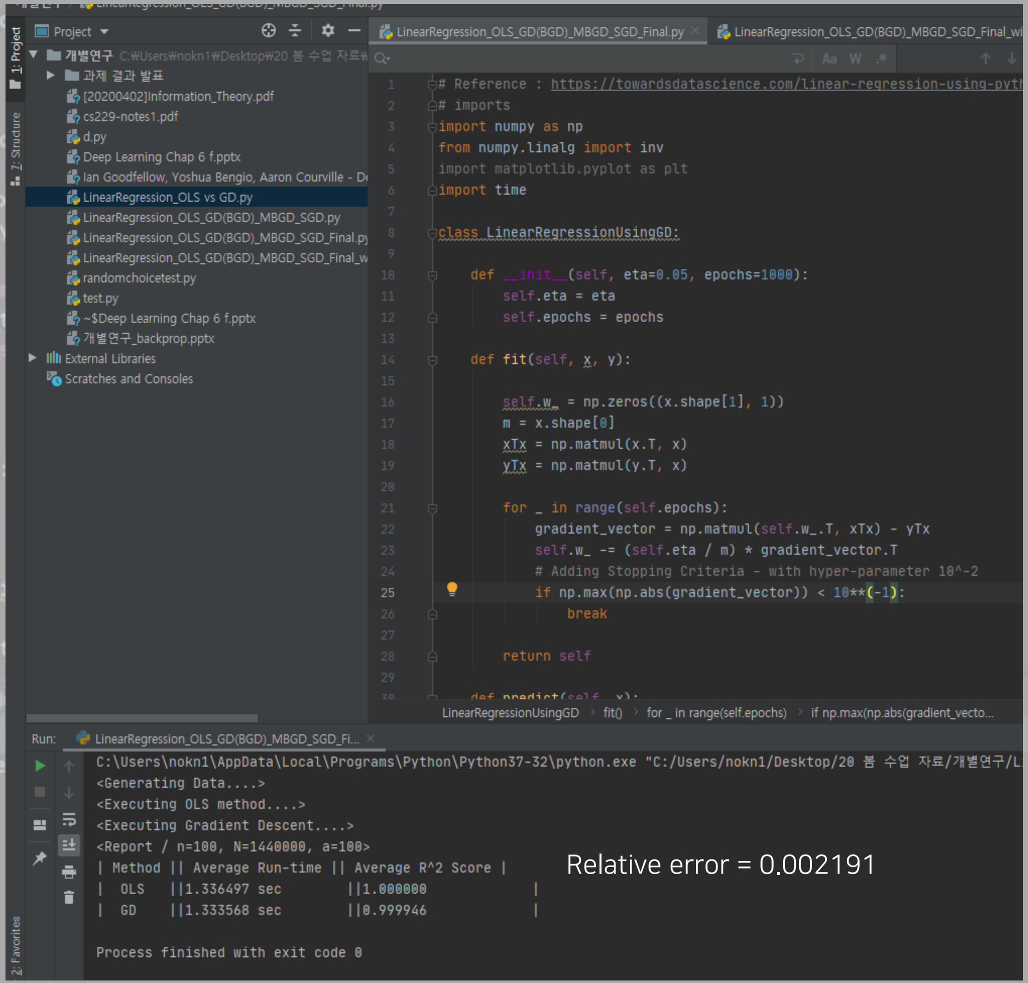
- Handling “TMI” Problem on GD case



Relative error = 0.082063

Stopping Criteria

- Handling “TMI” Problem on GD case



Relative error = 0.002191

"Why?" again

- Kind of Error Analysis?

1. Still we have "Formula Problem"
   - Maybe we can calculate gradient more conveniently.
   - And we can improve our algorithm; our algorithm of MBGD, SGD has serious problem.(Because of extracting a batch)
2. "TMI" Handling
   - We have to check that Stopping Criteria is valid; maybe we need more "time".
3. "Random Number Problem"
   - Randomly generated 'float' data causes MemoryError and kind of noises – for every "run", data differs, so that it might effect noises on our results.
4. "Laptop Problem" <=> "Second Problem"
   - Poor performance of my laptop regulates the size of the data, so we can't check our algorithms/calculations in Large data environment.
   - As a result of Laptop problem, each of calculations are done in just 2 seconds. This is also a big problem because of noises from randomly generated data

Reference

- Linear regression using matrix derivatives - Vivek Yadav's blog
URL : http://vxy10.github.io/2016/06/25/lin-reg-matrix/

- Linear Regression using Python - Animesh Agarwal
URL : https://towardsdatascience.com/linear-regression-using-python-b136c91bf0a2

이외 다수….

Special thanks to 김 아무개 군

Thank you.