R10946013 劉謦瑄

Homework assignments Week 12 (Students should submit their homework before 10 a.m. on December 15, 2021.)

3. Derivation of the Lipschitz constant for the logistic loss for regression estimation (2%)

Ans: (c)

Programming work

4. The gradient algorithm for logistic regression estimation (6%)

In this programming work we will build a gradient algorithm to find an estimate of regression coefficients in logistic regression model. To run such an algorithm, you need to:

1. Construct an iterative scheme based on the gradient algorithm.
2. Specify a **stepsize** for the iterative scheme. Here you are allowed to use whatever way you like to specify the **stepsize**.
3. Specify (a) a **stopping criterion**, (b) a **tolerance** for the error and (c) the **maximum number of iterations** for stopping the iterative scheme. Here you are allowed to use whatever way you like to specify the **stopping criterion** like the following one:

   Some measure on error $\leq$ tol OR The number of iterations $>$ max_iter.

   However the tolerance for the error and the maximum number of iterations should be

$$
\begin{aligned}
\text{tol} &= 5 \times 10^{-6}, \\
\text{max\_iter} &= 10,000.
\end{aligned}
$$

Data generation: We let the number of observations $n = 200$ and the number of covariates $p = 10$. We use the following model to generate the data:

$$
\begin{aligned}
\boldsymbol{\beta}^{\text{true}} &= (-1, 1, -1, 1, -1, 1, -1, 1, -1, 1), \\
\mathbf{x}_i &= (x_{i1}, x_{i2}, \cdots, x_{i10}), \\
x_{i1} &= 1 \text{ for } i = 1, 2, \cdots, 200, \\
x_{ij} &\sim \text{Normal}(0, 1) \text{ for } i = 1, 2, \cdots, 200 \text{ and } j = 2, 3, \cdots, 10, \\
y_i &\sim \text{Bernoulli}\left(\frac{\exp(\mathbf{x}_i^T \boldsymbol{\beta}^{\text{true}})}{1 + \exp(\mathbf{x}_i^T \boldsymbol{\beta}^{\text{true}})}\right) \text{ for } i = 1, 2, \cdots, 200.
\end{aligned}
$$

- 生成初始訓練與預測資料：(X 與 y)
  - X訓練資料：

```
import numpy as np
df = np.random.normal(0,1,(200,9)) #建立一個200*9陣列，內容為常態分佈的亂數值 平均0，標準差1
ones = np.ones((200,1) ,dtype=float) ## 第一直行的資料 (皆為1)
d = np.hstack((ones,df)) ## 合併上面兩個array
d
```

[13] ✓ 0.3s

```
... array([[ 1.        , -1.28579043, -0.92080805, ...,  0.21481575,
         1.422975  , -0.42182831],
       [ 1.        ,  1.4834412 , -1.74003085, ..., -1.69301932,
         0.41148259, -0.98027462],
       [ 1.        ,  1.51703484,  0.39417184, ..., -1.67434396,
        -0.13920761,  0.20002216],
       ...,
       [ 1.        ,  1.5053925 ,  0.48730986, ..., -1.03570155,
         1.12063432,  0.5888981 ],
       [ 1.        ,  0.34625933, -2.13407995, ...,  1.40676536,
        -0.37636965, -0.11023283],
       [ 1.        , -1.36226615,  0.13096245, ..., -1.034229  ,
         0.61139356, -0.46840271]])
```

  - Y 預測值：

```
import math
beta = [-1,1,-1,1,-1,1,-1,1,-1,1]
y = []
for i in range(200):
    y.append(  (math.exp(sum(d[i]*beta)) ) / ( 1+math.exp(sum(d[i]*beta)))  )

y_final = [] ## 建立 y 預測值 (Bernoulli分佈)
for i in y:
    if i>=0.5:
        y_final.append(1)
    else:
        y_final.append(0)

y_final_ = np.array(y_final)
```

[14] ✓ 0.6s

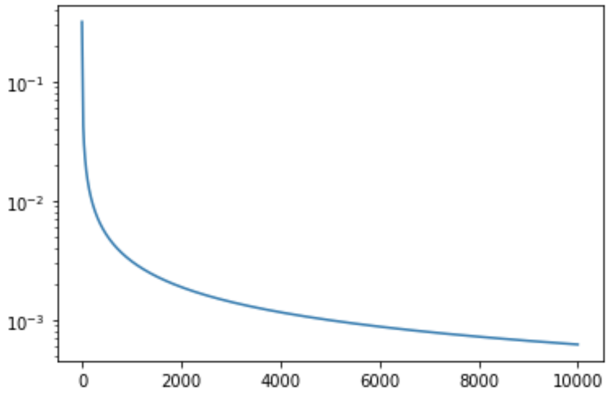
Tasks: Report line plots of the following two settings:

1. The x-axis is the number of iterations r and the y-axis is $||\nabla l(\boldsymbol{\beta}^r)||_2$, the Euclidean norm of the gradient of the loss function you use in your iterative scheme;

```
# 1 #log norm
import matplotlib.pyplot as plt
import seaborn as sn
%matplotlib inline
plt.yscale("log")
plt.plot( range(num_iterations), (norms))
# plt.plot( range(num_iterations), np.log(norms))
```

[166] ✓ 1.3s

```
[<matplotlib.lines.Line2D at 0x7fc765d61fd0>]
```



本圖有經過 log base with 10 處理後繪製。

2. The x-axis is the number of iterations r and the y-axis is $l(\boldsymbol{\beta}^r) - l(\boldsymbol{\beta}^*)$, the difference between the loss functions evaluated at the current update $\boldsymbol{\beta}^r$ and the *optimizer* $\boldsymbol{\beta}^*$. The optimizer $\boldsymbol{\beta}^*$ can be obtained from functions or software for carrying out logistic regression estimation available in your programming environment.

```python
#2 #loss
loss_diff = []
for i in range(len(loss)):
    loss_diff.append(loss[i]-loss_lg)

plt.plot( range(num_iterations), loss_diff )
```

[159] ✓ 1.4s