tags: 資料科學計算

Homework assignments Week 12 - Programming Work

Due date: 2021/12/18
Author: Hung Chun Hsu
Student ID: R10946017
線上閱讀: https://hackmd.io/@stupid-penguin/Sk0Ltscqt (https://hackmd.io/@stupid-penguin/Sk0Ltscqt)
Github Repository: https://github.com/stupidpenguin/Computation_of_Data_Science/blob/master/Homework_6/4_Gradient_Algorithm_for_Logistic_Regression_Estimation.ipynb (https://github.com/stupidpenguin/Computation_of_Data_Science/blob/master/Homework_6/4_Gradient_Algorithm_for_Logistic_Regression_Estimation.ipynb)

- For the coding details, please see the attached file -> "4_Gradient_Algorithm_for_Logistic_Regression_Estimation.ipynb", thanks!

Table of Contents

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

Derive the gradient of the loss function.

since $f_j = \vec{x}_j^T \beta$

$$\mathcal{L}(\beta) = \frac{1}{n}\sum_{j=1}^{n}\left[-y_j f_j + \log(1+e^{f_j})\right]$$

$$= \frac{1}{n}\sum_{j=1}^{n}\left[-y_j \vec{x}_j^T \beta + \log(1+e^{\vec{x}_j^T\beta})\right]$$

$(\nabla) \Rightarrow$

$$\nabla\mathcal{L}(\beta) = \frac{1}{n}\sum_{j=1}^{n}\left[-y_j \vec{x}_j^T + \frac{\exp(\vec{x}_j^T\beta)}{1+\exp(\vec{x}_j^T\beta)} \cdot \vec{x}_j^T\right]$$

Then the iterative schemes of gradient descent algorithms becomes :

$$\beta^{r+1} = \beta^r - c\nabla\mathcal{L}(\beta^r)$$

$$= \beta^r - c\left\{\frac{1}{n}\sum_{j=1}^{n}\left[-y_j \vec{x}_j^T + \frac{\exp(\vec{x}_j^T\beta)}{1+\exp(\vec{x}_j^T\beta)} \cdot \vec{x}_j^T\right]\right\}$$

with initial beta $\beta^0 = \vec{0}$.

4-1. Grdient Algorithm

- Construct an iterative scheme based on the gradient algorithm.

```
def Gradient_Descent(x, y, lr, p ,iterations, tol):  # c = lr (learning rate)
    # initial value of beta^0
    beta = np.zeros(10)

    loss_norm = []
    new_beta = []
    for j in range(iterations):
        gradient = np.zeros(p)
        for i in range(len(x)):
            first_term = -np.dot(y[i], x[i])
            common_factor = np.exp(np.dot(x[i], beta))
            second_term = np.dot( (common_factor)/(1 + common_factor),
                                  x[i])
            element = first_term + second_term
            gradient += element
        beta = beta - lr * (1/n) * (gradient)
        new_beta.append(beta)
        loss_norm.append(np.linalg.norm(gradient/n))

        # early stopping
        if np.linalg.norm(gradient/n) < tol:
            print(f' Early Stopping at iterations: {j}')
            return beta, np.array(new_beta), np.array(loss_norm)

    return beta, np.array(new_beta), np.array(loss_norm)
```

4-2. Stepsize

- Specify a stepsize for the iterative scheme. Here you are allowed to use whatever way you like to specify the stepsize.

> I choosed 1, 0.5, 0.1, 0.05 & 0.01 as stepsizes, then tested them seperately, and finally ploted the results of each at the last part.

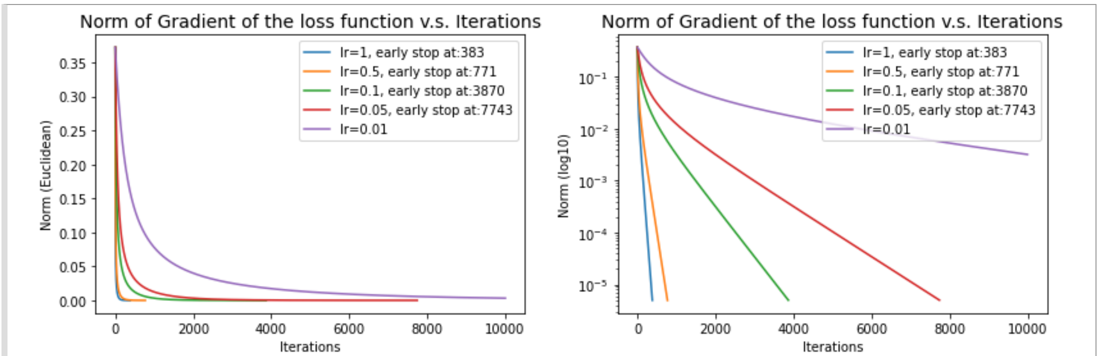
4-3. (a) Stopping criterion (b) Tolerance & © maximum number of iterations

> In my implementation, I make the algorithms stop whenever the conditions below are satisfied:
>
> 1. the current norm of loss is less than the tolerance ($= 5 \times 10^{-6}$)
> 2. number of interations reached max_iter ($= 10,000$)

Plot the result

(I) Number of Iterations v.s. Iteration Error

- plot the result with the y-scale: 1. eculidean norm 2. $\log_{10}$, respectively.



(II) Number of Iterations v.s Loss Difference

- plot the result with the y-scale: 1.linear 2. $\log_{10}$, respectively.