MATH 156S HOMEWORK 2

Due Apr. 14

Say we have training examples $(\boldsymbol{\phi}(\mathbf{x}_1), y_1), \ldots, (\boldsymbol{\phi}(\mathbf{x}_N), y_N)$ for binary classification. Recall that the logistic regression training is done by solving the following optimization problem

$$\hat{\mathbf{w}} = \operatorname{argmin}_{\mathbf{w}\in\mathbb{R}^p} \left[\ell_{LR}(\mathbf{w}) := \left(\sum_{i=1}^{N} \log\left(1 + \exp(\boldsymbol{\phi}(\mathbf{x}_i)^T \mathbf{w})\right) \right) - \mathbf{Y}^T \boldsymbol{\Phi}^T \mathbf{w} \right], \tag{1}$$

where

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \in \{0,1\}^N, \qquad \boldsymbol{\Phi} = \begin{bmatrix} \uparrow & & \uparrow \\ \boldsymbol{\phi}(\mathbf{x}_1) & \cdots & \boldsymbol{\phi}(\mathbf{x}_N) \\ \downarrow & & \downarrow \end{bmatrix} \in \mathbb{R}^{p\times N}. \tag{2}$$

Exercise 1 (Convexity of the logistic regression loss)**.** Let $\ell_{LR}(\mathbf{w})$ denote the logistic regression loss in (1). Let $\mathbf{Q} = \mathbf{Q}(\mathbf{w}) = [p_1, \ldots, p_N]^T \in \mathbb{R}^N$ where $p_i = \sigma(\boldsymbol{\phi}(\mathbf{x}_i)^T \mathbf{w})$ with $\sigma(x) = e^x/(1+e^x)$.

(i) Show that $\frac{\partial\sigma(x)}{x} = \sigma(x)(1-\sigma(x))$.

(ii) Show that

$$\frac{\partial \mathbf{Q}^T}{\partial \mathbf{w}} = \begin{bmatrix} \frac{\partial}{\partial w_1} \\ \vdots \\ \frac{\partial}{\partial w_p} \end{bmatrix} \begin{bmatrix} p_1 & \cdots & p_N \end{bmatrix} = \begin{bmatrix} \uparrow & & \uparrow \\ p_1(1-p_1)\boldsymbol{\phi}(\mathbf{x}_1) & \cdots & p_N(1-p_N)\boldsymbol{\phi}(\mathbf{x}_N) \\ \downarrow & & \downarrow \end{bmatrix} = \boldsymbol{\Phi}\mathbf{D}, \tag{3}$$

where $\mathbf{D}$ is the $N \times N$ diagonal matrix with diagonal entries $\mathbf{D}_{ii} = p_i(1-p_i)$.

(iii) Show that

$$\nabla_{\mathbf{w}} \ell_{LR}(\mathbf{w}) = \boldsymbol{\Phi}(\mathbf{Q} - \mathbf{Y}) \in \mathbb{R}^p, \qquad \mathbf{H} := \frac{\partial^2 \ell_{LR}(\mathbf{w})}{\partial \mathbf{w} \partial \mathbf{w}^T} = \nabla_{\mathbf{w}} \mathbf{Q}^T \boldsymbol{\Phi}^T = \boldsymbol{\Phi}\mathbf{D}\boldsymbol{\Phi}^T \in \mathbb{R}^{p\times p}. \tag{4}$$

(iv) Argue that the Hessian $\mathbf{H}$ in **(iii)** is positive semi-definite. Conclude that the logistic regression loss ℓ_{LR} is convex. (Hint: $\mathbf{D}$ is a diagonal matrix with positive diagonal entries.)

Exercise 2. The gradient descent (GD) algorithm for logistic regression training

$$\mathbf{w}_{t+1} \longleftarrow \mathbf{w}_t - \eta_t \nabla_{\mathbf{w}} \ell_{LR}(\mathbf{w}_t) = \mathbf{w}_t - \eta_t \mathbf{\Phi}(\mathbf{Q}(\mathbf{w}_t) - \mathbf{Y}), \tag{5}$$

is implemented as the python function 'fit_LR_GD' in the Jupyter notebook provided in the course repository.

```
def fit_LR_GD(Y, H, W0=None, sub_iter=100, stopping_diff=0.01):
        '''
        Convex optimization algorithm for Logistic Regression using Gradient Descent
        Y = (n x 1), H = (p x n) (\Phi in lecture note), W = (p x 1)
        Logistic Regression: Y ~ Bernoulli(Q), Q = sigmoid(H.T @ W)
        MLE -->
        Find \hat{W} = argmin_W ( sum_j ( log(1+exp(H_j.T @ W) ) - Y.T @ H.T @ W ) )
        '''
        if W0 is None:
            W0 = np.random.rand(H.shape[0],1) #If initial coefficients W0 is None, randomly initialize

        W1 = W0.copy()
        i = 0
        dist = 1
        grad = np.ones(W0.shape)
        while (i < sub_iter) and (np.linalg.norm(grad) > stopping_diff):
            Q = 1/(1+np.exp(-H.T @ W1))  # probability matrix, same shape as Y
            # grad = H @ (Q - Y).T + alpha * np.ones(W0.shape[1])
            grad = H @ (Q - Y)
            W1 = W1 - (np.log(i+1) / (((i + 1) ** (0.5)))) * grad
            i = i + 1
            # print('iter %i, grad_norm %f' %(i, np.linalg.norm(grad)))
        return W1
```

FIGURE 1. A python implementation of the gradient descent algorithm for logistic regression training (1).

(i) Modify the code (or make your own function) so that each gradient descent step (5), the current value of objective function ℓ_{LR} in (1) is printed. Test it on the MNIST example with digits $['0','1']$. Does the loss monotonically decrease?

(ii) The implemented version of GD for (5) uses the step size $\eta_t = \gamma \log(t+1)/\sqrt{(t+1)}$ with $\gamma = 1$. Test the same step sizes for $\gamma = 10$ and also $\eta_t = 1$ and $\eta_t = 1/t$. Test it on the MNIST example with digits $['0','1']$. Is there any difference in the way the loss function decrease?

(iii) If we use the step size of the form $\eta_t = \gamma t^{-\delta}$ for $0 \le \delta \le 1$, what would be your optimal choice of the hyperparameters γ and δ? (There is no answer and trying out a few choices and make your observation.)

Exercise 3. The *Newton-Ralphson algorithm* for logistic regression is an iterative convex optimization algorithm of the form

$$\mathbf{w}_{t+1} \longleftarrow \mathbf{w}_t - \mathbf{H}^{-1} \nabla_{\mathbf{w}} \ell_{LR}(\mathbf{w}_t), \tag{6}$$

where $\mathbf{H}$ is the Hessian of the logistic regression loss. See the Jupyter notebook provided in the course repository.

```
def fit_LR_NR(Y, H, W0=None, sub_iter=100, stopping_diff=0.01):
    '''
    Convex optimization algorithm for Logistic Regression using Newton-Ralphson algorithm.
    Y = (n x 1), H = (p x n) (\Phi in lecture note), W = (p x 1)
    Logistic Regression: Y ~ Bernoulli(Q), Q = sigmoid(H.T @ W)
    MLE -->
    Find \hat{W} = argmin_W ( sum_j ( log(1+exp(H_j.T @ W) ) - Y.T @ H.T @ W ) )
    '''
    ### Implement by yourself.
```

FIGURE 2. A python implementation of the Newton-Ralphson algorithm for logistic regression training (6) to be completed.

(i) Implement the Newton-Ralphson algorithm for logistric regression training in (6) in the same notebook, and reproduce Figure 2. How does the result compare with two optimization algorithms?

(ii) Compare with the gradient descent algorithm. Which one do you think is faster? Which one do you think is more efficient? (Do some experiments make your own argument. There is no single answer.)