第二次编程作业报告

——李昭阳 2021013445

一、 一元 Logistic 回归

1. 算法原理：

正向传播 $z = wx + b$ 且 $\hat{y} = \sigma(z) = \frac{1}{1+e^{-z}}$

反向传播：

记 Loss 函数 $L(\hat{y}, y) = -[ylog(\hat{y}) + (1-y)log(1-\hat{y})]$

$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$$\frac{\partial \hat{y}}{\partial z} = \hat{y}(1-\hat{y})$$

则有

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} = \left(-\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}\right) \cdot \hat{y}(1-\hat{y}) = \hat{y} - y$$

故

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w} = (\hat{y} - y) \cdot x$$

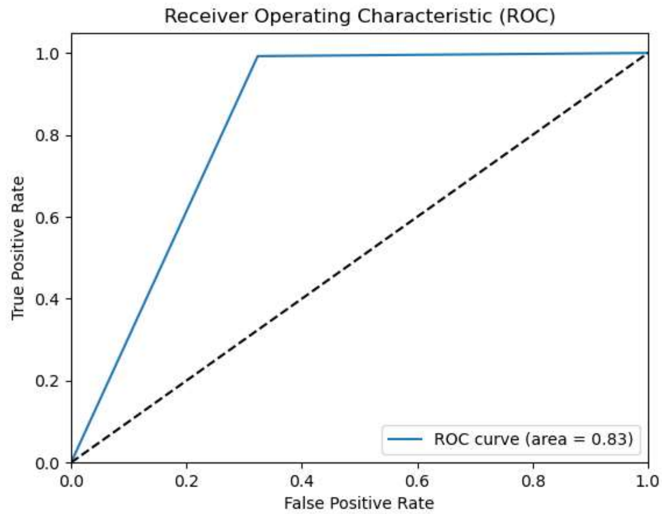
$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial b} = \hat{y} - y$$

2. 代码实现：

```python
def train_logistic_regression(X, y, learning_rate=0.005, num_iterations=100000):
    m, n = X.shape
    w = np.zeros(n)
    b = 0

    for i in range(num_iterations):
        idx = np.random.randint(m)
        x_i = X[idx]
        y_i = y[idx]

        # 计算预测值
        z = np.dot(x_i, w) + b
        h = sigmoid(z)

        # 计算梯度
        gradient_w = (h - y_i) * x_i
        gradient_b = h - y_i

        # 更新参数
        w -= learning_rate * gradient_w
        b -= learning_rate * gradient_b

    return w, b
```

3. 算法效果：

```
准确率 (Accuracy): 0.84
精确率 (Precision): 0.76
召回率 (Recall): 0.99
F1-score: 0.86
auROC: 0.83
```



二、 Softmax 回归

1. 算法原理：

正向传播 $z = Wx + b$ 且 $\widehat{y_k} = \sigma(z) = \frac{e^{-z_k}}{\sum e^{-z_j}}$

反向传播:

记 Loss 函数 $L(\hat{y}, y) = -\sum_{k=1}^{K} y_k log(\widehat{y_k})$

$$\frac{\partial L}{\partial \hat{y}_k} = \sum_{k=1}^{K} -\frac{y_k}{\hat{y}_k}$$

$$\frac{\partial \widehat{y_k}}{\partial z_j} = \hat{y}_k(\delta_{kj} - \hat{y})$$

其中 $\delta_{kj} = \begin{cases} 1, & k = j \\ 0, & else \end{cases}$

则有

$$\frac{\partial L}{\partial z_j} = \sum_{k=1}^{K} \frac{\partial L}{\partial \widehat{y_k}} \cdot \frac{\partial \widehat{y_k}}{\partial z_j} = \sum_{k=1}^{K} -\frac{y_k}{\hat{y}_k} \cdot \hat{y}_k(\delta_{kj} - \hat{y})$$

$$= \sum_{k=1}^{K} y_k(\delta_{kj} - \hat{y}) = -y_j + \hat{y}_j$$

故

$$\frac{\partial L}{\partial w_k}=\frac{\partial L}{\partial z_k}\cdot\frac{\partial z_k}{\partial w_k}=(\hat{y}-y)\cdot x$$

$$\frac{\partial L}{\partial b_k}=\frac{\partial L}{\partial z_k}\cdot\frac{\partial z_k}{\partial b_k}=\hat{y}-y$$

2. 代码实现：

```python
def train_softmax_regression(X, y, learning_rate=0.005, num_iterations=100000):
    y = to_one_hot(y, num_classes=np.max(y) + 1)
    m, n = X.shape
    K = y.shape[1]  # 类别数量
    W = np.zeros((n, K))
    b = np.zeros(K)

    for i in range(num_iterations):
        idx = np.random.randint(m)
        x_i = X[idx]
        y_i = y[idx]

        # 计算预测值
        z = np.dot(x_i, W) + b
        h = softmax(z)

        # 计算梯度
        gradient_W = np.outer(x_i, (h - y_i))
        gradient_b = h - y_i

        # 更新参数
        W -= learning_rate * gradient_W
        b -= learning_rate * gradient_b

    return W, b
```

3. 算法效果：

```
准确率 (Accuracy): 0.83
精确率 (Precision): 0.83
召回率 (Recall): 0.83
F1-score: 0.82
auROC: 0.91
confusion_matrix:
 [[608   0   0   2   1  10   3   4   1   1]
 [  4 716   2   5   5   2   3   2   6   3]
 [ 29  29 504  32  15   8  10  10  44  10]
 [ 13   6   5 558   7  72   1  12  20   5]
 [  8   1   1   0 614   3   5   2   5  33]
 [ 23   3   0  13  24 493   4   7  24   5]
 [ 45   4   6   0  27  33 510   4   9   1]
 [  2   2   6  11  11   5   0 609  11  41]
 [ 17  22   4  30  43  77  10  10 426  21]
 [ 12   3   2   3 107  13   0  17   2 501]]
```