1 数据预处理

这里直接使用了助教帮我们处理好的数据，训练数据为（54256，510），测试数据为（27622，510）。

```
def data_process(train_x, train_y, test_x):
    """数据预处理"""
    x_train = np.array(train_x.iloc[:, 1:])
    y_train = np.array(train_y.iloc[:, 1:])
    x_test = np.array(test_x.iloc[:, 1:])
    return x_train, y_train, x_test


def normalize(X, train=True, X_mean=None, X_std=None):
    """数据标准化"""
    if train:
        # 如果是训练集，需要计算平均值和标准差，如果不是，直接使用计算好的
        X_mean = np.mean(X, axis=0).reshape(1, -1)
        X_std = np.std(X, axis=0).reshape(1, -1)

    X = (X - X_mean) / (X_std + 1e-8) # 防止除0
    return X, X_mean, X_std


def split_data(x_train, y_train, radio):
    """按照radio划分训练集和验证集"""
    size = len(x_train)
    X_train = x_train[: math.floor(radio * size), :]
    Y_train = y_train[: math.floor(radio * size), :]
    X_val = x_train[math.floor(radio * size): , :]
    Y_val = y_train[math.floor(radio * size): , :]
    return X_train, Y_train, X_val, Y_val
```

2 模型建立

2.1逻辑回归模型

这里采用逻辑回归的方式，逻辑回归其实就是之前的线性回归方程再加一个sigmoid函数，将输出值映射到[0, 1]范围。

$$z = \sum_{i=1}^{n} w_i x_i + b$$

其中x_i为第i个特征，w_i为其对应的权重，b为偏置项，y为预测结果。

$$f_{w,b}(x) = \sigma(z) = \frac{1}{1+e^{-z}}$$

z即为从线性回归方程中求出的函数值，然后代入sigmoid函数，得出$f_{w,b}$。

2.2 损失函数

在线性回归中我们一般使用预测值和label之间的平均欧式距离来测量预测误差，损失函数为所有预测误差的平均值。但是在分类问题中，模型一般针对各类别输出一个概率分布，因此常用**交叉熵**作为损失函数。

交叉熵可用于衡量两个概率分布之间的相似性，两个概率分布越相似，则交叉熵越小；反之，越大。

交叉熵公式为：

$$H(p,q) = -\sum_{x} p(x)\ln(q(x))$$

代入本实验中：

$$Loss = \sum_{n} C(f(x^n), \hat{y}^n)$$

其中$f(x^n)$为预测结果，$\hat{y}^n$为真实结果。

$$C(f(x^n), \hat{y}^n) = -\left[\hat{y}^n \ln f(x^n) + (1-\hat{y}^n)\ln(1-f(x^n))\right]$$

2.3 梯度下降

根据梯度下降公式：

$$w_i = w_{i-1} - \eta_w grad_w = w_i - \eta_w \frac{\partial Loss}{\partial w}$$

$$b_i = b_{i-1} - \eta_b grad_b = b_{i-1} - \eta_b \frac{\partial Loss}{\partial b}$$

其中η_w为权重w更新时的学习率，η_b为权重b更新时的学习率。

因此我们需要计算$\frac{\partial Loss}{\partial w}$和$\frac{\partial Loss}{\partial b}$，即$w$和$b$的梯度：

$$grad_w = \frac{\partial Loss}{\partial w} = \sum_{n} -(\hat{y}^n - f_{w,b}(x^n))x_i^n$$

$$grad_b = \frac{\partial Loss}{\partial b} = \sum_n -(\hat{y}^n - f_{w,b}(x^n))$$

推导过程如下：

由上可知，$LossFunction$如下所示：

$$Loss_{w,b} = \sum_n -\left[\hat{y}^n \ln f_{w,b}(x^n) + (1-\hat{y}^n)\ln(1-f_{w,b}(x^n))\right]$$

因此，计算$\frac{\partial Loss}{\partial w}$只需计算$\frac{\partial \ln f_{w,b}(x^n)}{\partial w}$和$\frac{\partial \ln(1-f_{w,b}(x^n))}{\partial w}$。

其中：

$$f_{w,b}(x) = \sigma(z) = \frac{1}{1+e^{-z}}$$

$$z = \sum_n w_i x_i + b$$

因此：

$$\frac{\partial \ln f_{w,b}(x^n)}{\partial w} = \frac{\partial \ln f_{w,b}(x^n)}{\partial z} \cdot \frac{\partial z}{\partial w}$$

$$\frac{\partial \ln(1-f_{w,b}(x^n))}{\partial w} = \frac{\partial \ln(1-f_{w,b}(x^n))}{\partial z} \cdot \frac{\partial z}{\partial w}$$

其中易得：

$$\frac{\partial z}{\partial w} = x_i$$

所以现在只需要计算$\frac{\partial \ln f_{w,b}(x^n)}{\partial z}$和$\frac{\partial \ln(1-f_{w,b}(x^n))}{\partial z}$：

$$\frac{\partial \ln f_{w,b}(x^n)}{\partial z} = \frac{\partial \ln(\frac{1}{1+e^{-z}})}{\partial z} = (1+e^{-z}) \cdot \frac{1}{(1+e^{-z})^2} \cdot e^{-z} = \frac{e^{-z}}{1+e^{-z}} = 1-\sigma(z)$$

$$\frac{\partial \ln(1-f_{w,b}(x^n))}{\partial z} = \frac{\partial \ln(\frac{e^{-z}}{1+e^{-z}})}{\partial z} = \frac{1+e^{-z}}{e^{-z}} \cdot \frac{-e^{-z}(1+e^{-z}) + e^{-z} \cdot e^{-z}}{(1+e^{-z})^2} = \frac{-1}{1+e^{-z}} = -\sigma(z)$$

由此两个偏微分都算好了，只需要将结果代入$\frac{\partial Loss}{\partial w}$：

$$\frac{\partial Loss}{\partial w} = \sum_n -\left[\hat{y}^n(1-f_{w,b}(x^n))x_i^n + (1-\hat{y}^n)(-f_{w,b}(x^n))x_i^n\right] = \sum_n -(\hat{y}^n - f_{w,b}(x^n))x_i^n$$

同理可求得$\frac{\partial Loss}{\partial b}$。

```
def shuffle(x, y):
    """打乱数据"""
    random_list = np.arange(len(x))
    np.random.shuffle(random_list)
    return x[random_list], y[random_list]

def sigmoid(z):
    """sigmoid函数"""
    # 避免溢出，如果sigmoid函数的最小值比1e-8小，只会输出1e-8；而比1 - (1e-8)大，则只输出
1 - (1e-8)
    return np.clip(1 / (1.0 + np.exp(-z)), 1e-8, 1-(1e-8))


def f(x, w, b):
    """返回y：即概率值"""
    return sigmoid(np.dot(x, w) + b)

def gradient(x, y, w, b):
    """计算梯度"""
    y_pred = f(x, w, b)
    w_grad = np.dot(x.T, y_pred - y)
    b_grad = np.sum(y_pred - y)
    return w_grad, b_grad

def cal_accuracy(y_pred, y):
    """计算正确率"""
    acc = 1 - np.mean(np.abs(y_pred - y))
    return acc

def cal_cross_entropy(y_pred , y):
    """计算交叉熵"""
    loss = -np.dot(y.reshape(1, -1), np.log(y_pred)) - np.dot((1 -
y).reshape(1, -1), np.log(1 - y_pred))
    return loss[0][0]
```

2.4 学习率更新

本实验采用经典的Adagrad算法更新学习率，该算法的思想是独立地适应模型的每个参数：具有较大偏导的参数相应有一个较大的学习率，而具有小偏导的参数则对应一个较小的学习率。因此更新后的学习率为：

$$\eta_{new} = \frac{\eta}{\sqrt{\sum_{i=0}^{t-1}(grad_i)^2}}$$

具体来说，每个参数的学习率会缩放各参数反比于其历史梯度平方值总和的平方根，因此Adagrad公式为：

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\sum_{i=0}^{t-1}(grad_i)^2}} grad_{t-1}$$

其中$\sum_{i=0}^{t-1}(grad_i)^2$为之前所有梯度的平方和。

3 模型训练与验证

```
def train_model(x_train, y_train, x_val, y_val):
    """建立模型"""
    w = np.ones((x_train.shape[1], 1)) # weight
    b = 1 # bias
    epochs = 30 # 迭代次数
    batch_size = 15 # 每个批次的样本数
    lr = 0.05 # 学习率

    train_loss = [] # 训练集损失
    val_loss = []
    train_acc = []# 训练集正确率
    val_acc = []

    step = 1
    for epoch in range(epochs):
        x_train, y_train = shuffle(x_train, y_train)
        for i in range(int(len(x_train / batch_size))):
            x = x_train[i*batch_size: (i+1)*batch_size, :]
            y = y_train[i*batch_size: (i+1)*batch_size, :]

            w_grad, b_grad = gradient(x, y, w, b)

            w -= (lr / step**0.5) * w_grad
            b -= (lr / step**0.5) * b_grad

            step += 1


        y_train_pred = f(x_train, w, b)
        train_acc.append(cal_accuracy(np.round(y_train_pred), y_train))
        train_loss.append(cal_cross_entropy(y_train_pred, y_train) /
len(x_train))

        y_val_pred = f(x_val, w, b)
        val_acc.append(cal_accuracy(np.round(y_val_pred), y_val))
        val_loss.append(cal_cross_entropy(y_val_pred, y_val) / len(x_val))


    print('训练集正确率: ' + str(train_acc[-1]))
    print('训练集误差: ' + str(train_loss[-1]))
```

```
    print('验证集正确率: ' + str(val_acc[-1]))
    print('验证集误差: ' + str(val_loss[-1]))

    return w, b, train_acc, train_loss, val_acc, val_loss


def plot_acc_loss(train_acc, train_loss, val_acc, val_loss):
    """画图"""
    plt.plot(train_loss)
    plt.plot(val_loss)
    plt.title('Loss')
    plt.legend(['train', 'val'])
    plt.show()

    plt.plot(train_acc)
    plt.plot(val_acc)
    plt.title('Acc')
    plt.legend(['train', 'val'])
    plt.show()
```

4 模型测试

```
def test_model(x_test, w, b):
    """测试模型"""
    y_test = f(x_test, w, b)
    y_test = np.round(y_test)
    # 保存为结果文件
    with open('result.csv', mode='w', newline='') as result:
        csv_writer = csv.writer(result)
        header = ['id', 'label']
        csv_writer.writerow(header)
        for i in range(len(y_test)):
            row = [str(i), int(y_test[i][0])]
            csv_writer.writerow(row)
```