1 数据预处理

本实验没有用到所有的数据，仅用到观测指标为PM2.5的数据。

2014/1/1	豐原	PM2.5	26	39	36	35	31	28	25	20	19

所以提取数据的时候只需要取出每一天的该项指标的数据即可，同时为了更好的训练，将连续9个小时的数据作为训练集，第10个小时的数据作为标签，每天有24个小时，因此每天可以划分出15份训练集和标签。

```
# 数据预处理
def data_process(data):
    # 取24个小时的观测值
    data = data.iloc[:, 3: ]
    # 将空数据填充为0
    data = data.replace(['NR'], [0.0])
    array = np.array(data, dtype=float)

    # x为训练集列表，y为标签列表
    x = []
    y = []

    for i in range(0, 4320, 18):
        # 将连续9个小时的数据作为训练集，第10个小时的数据作为标签，每天有24个小时，因此每天可以划分出15份训练集和标签
        for j in range(24-9):
            tmp_x = array[i+9, j: j+9]
            tmp_y = array[i+9, j+9]
            x.append(tmp_x)
            y.append(tmp_y)

    # 此时shape(X) = (4320/18*15, 9) = (3600, 9)
    X = np.array(x, dtype=float)
    Y = np.array(y, dtype=float)

    return X,Y
```

2 模型建立

2.1 回归模型

采用最简单的线性回归。

$$y = \sum_{i=0}^{8} w_i x_i + b$$

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。

2.2 损失函数

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，其中乘1/2是为了后续求导方便。

$$Loss_{label} = \frac{1}{2n} \sum_{i=0}^{n-1} (\hat{y}^i - y^i)^2$$

其中$\hat{y}^i$为第i个label，y^i为第i个预测值，n为参加训练的样本个数。

为防止过拟合，加入正则项，此处采用L2正则项：

$$Loss_{regularization} = \frac{1}{2} \sum_{i=0}^{8} w_i^2$$

$$Loss = Loss_{label} + \beta Loss_{regularization} = \frac{1}{2} \left[\frac{1}{n} \sum_{i=0}^{n-1} (\hat{y}^i - y^i)^2 + \beta \sum_{i=0}^{8} w_i^2 \right]$$

其中β为正则项系数。

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：

$$grad_w = \frac{\partial Loss}{\partial w} = \frac{1}{n} \sum_{i=0}^{n-1} (\hat{y}^i - \sum_{j=0}^{8} w_j x_j - b) \cdot (-x_j) + \beta \sum_{i=0}^{8} w_i$$

$$grad_b = \frac{\partial Loss}{\partial b} = \frac{1}{n} \sum_{i=0}^{n-1} (\hat{y}^i - \sum_{j=0}^{8} w_j x_j - b) \cdot (-1)$$

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。

```
# 建立模型
def build_model(train_x, train_y, epochs):
    bias = 0 # 偏置项
    weights = np.ones(9) # 权重
    bg_sum = 0 # 偏置项的梯度平方和
    wg_sum = np.zeros(9) # 权重的梯度平方和
    lr = 1 # 学习率
    reg_rate = 0.001 # 正则项系数

    for epoch in range(epochs):
        b_grad = 0
        w_grad = np.zeros(9)
        for i in range(3200):
            # 由于只需要观测值表为pm2.5的数据，所以第二维为9
            b_grad += (train_y[i] - weights.dot(train_x[i, :]) - bias) * (-1)
            for j in range(9):
                w_grad[j] += (train_y[i] - weights.dot(train_x[i, :]) - bias) *
(-train_x[i, j])

        # 将梯度求平均
        b_grad /= 3200
        w_grad /= 3200

        # 加上正则项
        for i in range(9):
            w_grad[i] += reg_rate * weights[i]

        # adgrad
```

```
        bg_sum += b_grad**2
        wg_sum += w_grad**2

        # 更新权重和偏置项
        bias -= (lr / bg_sum**0.5) * b_grad
        weights -= (lr / wg_sum**0.5) * w_grad


        # 每训练200轮，输出一次Loss
        if epoch % 200 == 0:
            loss = 0
            for i in range(3200):
                loss += (train_y[i] - weights.dot(train_x[i, :]) - bias)**2
            print('after {} epochs, the loss on train data is:'.format(epoch),
loss/3200)

    return weights, bias
```

3 模型验证

本实验简单地将数据分为8：1，其中3200条为训练集，400条为验证集，如果想要更好的效果，可以采用n折交叉验证。

```
# 验证模型
def val_model(val_x, val_y, weights, bias):
    loss = 0
    for i in range(400):
        loss += (val_y[i] - weights.dot(val_x[i, :]) - bias)**2
    return loss / 400
```

4 模型测试

数据处理部分和对训练集的处理差不多，然后将测试数据代入线性回归方程，输出结果。

```
def test_data_process(data):
    data = data.iloc[:, 2: ]
    # 将空数据填充为0
    data = data.replace(['NR'], [0.0])
    array = np.array(data, dtype=float)

    x = []
```

```
    for i in range(0, 4320, 18):
        tmp_x = array[i+9, :]
        x.append(tmp_x)

    X = np.array(x, dtype=float)
    return X
```

```
def test(test_x, weights, bias):
    y = np.dot(test_x, weights)
    return y
```