手搓机器学习系列——线性回归模型（基于梯度下降算法）

机器学习系列开更！【机器学习系列】将陆续更新常用机器学习模型的数学理论介绍以及案例实战，代码原创且全部公开（配操作指南）。

尽管在matlab和python中有很多现成的关于机器学习相关模型的包，但是为了更好学习掌握这些模型，博主将遵循"少调包，多手搓，打造自己的机器学习模型代码库"。

`线性回归`是一种用于建立一个或者多个自变量与一个因变量之间线性关系的统计模型，`梯度下降法`是求解该模型参数的常用优化算法。本篇讲解的是基于梯度下降法的多元线性回归的数学理论及应用。

---

1. 线性回归模型的数学表达

设自变量有 n 个，分别为 $x_1, x_2, \cdots, x_n$，因变量为 y。多元线性回归模型的表达式为：

$$\hat{y} = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_n x_n = \sum_{i=0}^{n} \omega_i x_i \tag{1}$$

其中，$\hat{y}$ 是模型的预测值，ω_i（$i = 0, 1, \cdots, n$）是待估计的模型参数，为了统一形式，引入 $x_0 = 1$。对于给定的数据集 $\{(x^{(j)}, y^{(j)})\}_{j=1}^{m}$，其中 $x^{(j)} = (x_0^{(j)}, x_1^{(j)}, \cdots, x_n^{(j)})$ 是第 j 个样本的自变量向量，$y^{(j)}$ 是第 j 个样本的因变量值。

---

2. 数据标准化

举一个梨子来说明数据标准化的重要性。

假如你要分析梨子的两个特征：重量（单位：克）和直径（单位：厘米），来预测梨子的甜度。现有三个梨子，其重量分别是150克、200克、250克，直径分别是6厘米、8厘米、10厘米。在构建预测模型时，重量的数值范围较大，直径的数值范围相对较小。如果不进行数据标准化，模型可能会过度关注重量这个特征，而忽略直径的影响，因为重量数值的波动幅度更大。进行数据标准化后，这两个特征的数据会被转换到相同的尺度，模型就能平等地对待重量和直径这两个特征，综合它们的信息来更准确地预测梨子的甜度，提升模型的性能和泛化能力。

- **Z-score 标准化公式**

对于数据点 x_i，其标准化后的Z值定义为：

$$Z_i = \frac{x_i - \mu}{\sigma} \tag{2}$$

其中，$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$ 为数据集的均值，$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$ 为数据集的标准差。

- **统计特性推导**

1. 均值为0

$$\mathbb{E}[Z] = \mathbb{E}\left[\frac{x_i-\mu}{\sigma}\right] = \frac{1}{\sigma}(\mathbb{E}[x_i] - \mu) = \frac{1}{\sigma}(\mu - \mu) = 0$$

2. 方差为1

$$\mathrm{Var}(Z) = \mathbb{E}\left[\left(\frac{x_i-\mu}{\sigma}\right)^2\right] = \frac{1}{\sigma^2}\mathbb{E}\left[(x_i-\mu)^2\right] = \frac{\sigma^2}{\sigma^2} = 1$$

- **参数还原推导**

标准化后的回归模型：

$$\hat{y} = w_0' + \sum_{i=1}^{n} w_i' x_i' \tag{3}$$

将标准化公式代入后展开：

$$\begin{aligned} \hat{y} &= w_0' + \sum_{i=1}^{n} w_i' \left(\frac{x_i - \mu_i}{\sigma_i}\right) \\ &= w_0' - \sum_{i=1}^{n} \frac{w_i'\mu_i}{\sigma_i} + \sum_{i=1}^{n} \frac{w_i'}{\sigma_i} x_i \end{aligned} \tag{4}$$

原始特征的回归模型：

$$\hat{y} = w_0 + \sum_{i=1}^{n} w_i x_i \tag{5}$$

则参数对应关系：

$$\begin{cases} w_0 = w_0' - \sum_{i=1}^{n} \frac{w_i'\mu_i}{\sigma_i} \\ w_i = \frac{w_i'}{\sigma_i} \quad (i = 1, 2, \ldots, n) \end{cases} \tag{6}$$

3. 损失函数

为了衡量模型预测值与真实值之间的差异，定义损失函数（这里使用均方误差，MSE）：

$$J(\omega) = \frac{1}{2m}\sum_{j=1}^{m}(\hat{y}^{(j)} - y^{(j)})^2 = \frac{1}{2m}\sum_{j=1}^{m}\left(\sum_{i=0}^{n}\omega_i x_i^{(j)} - y^{(j)}\right)^2 \tag{7}$$

其中，m 是样本数量，$J(\omega)$ 表示所有样本的预测误差的平方和的平均值的一半，除以2是为了后续偏导计算形式上的美观。

---

4. 梯度下降法

梯度下降法的目标是找到一个函数的极小值点。其核心思想：从一个初始点出发，沿着函数在该点的梯度的反方向，因为**梯度方向是函数增长最快的方向，所以其反方向就是函数下降最快的方向**，以一定的步长移动到一个新的点，不断重复这个过程，直到满足停止条件（一般是函数值的变化小于某个阈值，或者达到最大迭代次数等）。

针对上面的线性回归模型，梯度下降法的目标就是要通过迭代更新参数ω，使得损失函数$J(\omega)$ 最小化。参数更新的公式为：

$$\omega_i := \omega_i - \alpha \frac{\partial J(\omega)}{\partial \omega_i} \tag{8}$$

其中，α 是学习率（步长），控制每次参数更新的幅度；$\frac{\partial J(\omega)}{\partial \omega_i}$ 是损失函数 $J(\omega)$ 关于参数 ω_i 的偏导数。 计算偏导数：

$$\frac{\partial J(\omega)}{\partial \omega_i} = \frac{1}{m} \sum_{j=1}^{m} \left(\sum_{i=0}^{n} \omega_i x_i^{(j)} - y^{(j)} \right) x_i^{(j)} \tag{9}$$

在每次迭代中，对所有的参数 ω_i （$i = 0, 1, \cdots, n$）同时进行更新，当损失函数的变化小于某个阈值或达到最大迭代次数时，得到最终的参数值。

---

5. 项目实战——一个四元线性回归案例

数据集Folds5x2_pp.csv共有9568个样本数据，每个数据有5列，分别是:AT（温度），V（压力），AP（湿度），RH（压强），PE（输出电力)。建立四元线性回归模型，AT（温度），V（压力），AP（湿度），RH（压强）为自变量，PE（输出电力)为应变量。

- **导入必要的包**

```
# 导入必要的包
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
```

- **数据导入函数**

```
# 导入数据并对作标准化：提取文件中的每一列元素
def load_data(file_path):

    df = pd.read_csv(file_path)

    # 提取特征
    features = df[['AT', 'V', 'AP', 'RH']]
    # 原始数据:后面参数返回会用到
    x1 = np.array(features['AT'])
    x2 = np.array(features['V'])
    x3 = np.array(features['AP'])
    x4 = np.array(features['RH'])
    # 标准化
    features = (features - features.mean()) / features.std()
    x_1 = np.array(features['AT'])
    x_2 = np.array(features['V'])
    x_3 = np.array(features['AP'])
    x_4 = np.array(features['RH'])
    y = np.array(df['PE'])

    return x1, x2, x3, x4, x_1, x_2, x_3, x_4, y
```

- **定义四元线性回归模型**

```
# 定义线性回归模型
def y_predict(x_1, x_2, x_3, x_4, w_0, w_1, w_2, w_3, w_4):
    y = w_0 + w_1 * x_1 + w_2 * x_2 + w_3 * x_3 + w_4 * x_4  # 对应公式(1)
    return y
```

- **定义损失函数**

```
# 定义损失函数（MSE）
def loss(x_1, x_2, x_3, x_4, w_0, w_1, w_2, w_3, w_4, y):
    m = len(y)
    predictions = y_predict(x_1, x_2, x_3, x_4, w_0, w_1, w_2, w_3, w_4)
    losses = (1 / (2 * m)) * np.sum((predictions - y) ** 2)  # 对应公式(2)
    return losses
```

- **梯度下降法**

```
# 梯度下降法
def gradient_descent(x_1, x_2, x_3, x_4, w_0, w_1, w_2, w_3, w_4, y, alpha,
epoch):
    m = len(y)
    cost_history = []
    for i in range(1, epoch + 1):
        predictions = y_predict(x_1, x_2, x_3, x_4, w_0, w_1, w_2, w_3, w_4)
        # 对应公式(4)
        w_0_derivative = (1 / m) * np.sum(predictions - y)
        w_1_derivative = (1 / m) * np.sum((predictions - y) * x_1)
        w_2_derivative = (1 / m) * np.sum((predictions - y) * x_2)
        w_3_derivative = (1 / m) * np.sum((predictions - y) * x_3)
        w_4_derivative = (1 / m) * np.sum((predictions - y) * x_4)
        # 对应公式(3)
        w_0 = w_0 - alpha * w_0_derivative
        w_1 = w_1 - alpha * w_1_derivative
        w_2 = w_2 - alpha * w_2_derivative
        w_3 = w_3 - alpha * w_3_derivative
        w_4 = w_4 - alpha * w_4_derivative
        cost = loss(x_1, x_2, x_3, x_4, w_0, w_1, w_2, w_3, w_4, y)
        cost_history.append(cost)
        print(f"第{i}次迭代，w_0：{w_0}  w_1：{w_1}  w_2：{w_2}  w_3：{w_3}  w_4：
{w_4}")
    return w_0, w_1, w_2, w_3, w_4, cost_history
```

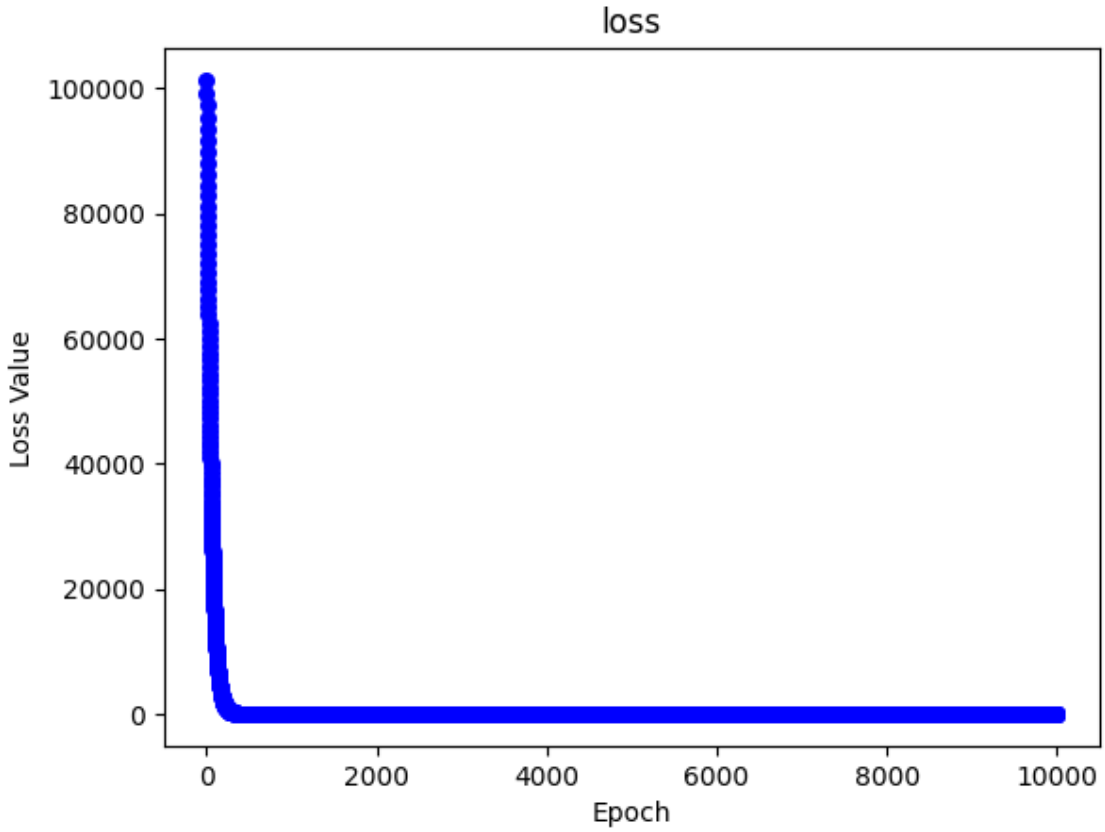
- **主函数**

```
# 主函数
def main():
    # 数据准备
    file_path = "C:/Users/Administrator/Desktop/pytorchlearning/Folds5x2_pp.csv"
 #此处更换为自己的数据集
    x1, x2, x3, x4, x_1, x_2, x_3, x_4, y = load_data(file_path)

    # 统计量计算
    mean_1 = np.mean(x1)
    mean_2 = np.mean(x2)
    mean_3 = np.mean(x3)
    mean_4 = np.mean(x4)
```

```
    std_1 = np.std(x1)
    std_2 = np.std(x2)
    std_3 = np.std(x3)
    std_4 = np.std(x4)

    # 初参数始化
    w_0 = 0
    w_1 = 0
    w_2 = 0
    w_3 = 0
    w_4 = 0
    epoch = 10000
    alpha = 0.01

    # 得到最终参数值并打印
    w_0, w_1, w_2, w_3, w_4, cost_history = gradient_descent(x_1, x_2, x_3, x_4,
w_0, w_1, w_2, w_3, w_4, y, alpha, epoch)

    print(f"最终的标准化处理下的参数w_0：{w_0},最终的w_1：{w_1},最终的w_2：{w_2},最终的
w_3：{w_3},最终的w_4：{w_4}")

    # 参数转换，公式(6)
    w_0_end = w_0 - w_1 * mean_1 / std_1 - w_2 * mean_2 / std_2 - w_3 * mean_3 /
std_3 - w_4 * mean_4 / std_4
    w_1_end = w_1 / std_1
    w_2_end = w_2 / std_2
    w_3_end = w_3 / std_3
    w_4_end = w_4 / std_4

    print(f"最终的返回参数w_0_end：{w_0_end},最终的w_1_end：{w_1_end},最终的w_2_end：
{w_2_end},最终的w_3_end：{w_3_end},最终的w_4_end：{w_4_end}")

    # 绘制损失值散点图
    x = list(range(1, len(cost_history) + 1))
    plt.scatter(x, cost_history, c='b', s=30)
    plt.title('loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss Value')
    plt.show()


if __name__ == "__main__":
    main()
```

- 运行结果

```
最终的标准化处理下的参数w_0：454.3650094063517,最终的w_1：-14.73708982181992,最终的
w_2：-2.9727874959734817,最终的w_3：0.3687486844615209,最终的
w_4：-2.3075431409815446
最终的返回参数w_0_end：454.60019439331427,最终的w_1_end：-1.9775797311790506,最终的
w_2_end：-0.2339445932587303,最终的w_3_end：0.06209486220299702,最终的
w_4_end：-0.15805625056488867
```



6. 适用于任意维线性回归模型的纯手搓代码

6.1 全部代码

```
# 导入必要的包
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
matplotlib.use('TkAgg')


# 数据加载与标准化
def load_data(file_path, y_column):
    """
    加载数据并进行标准化处理
    :param file_path: 数据文件路径
    :param y_column: 目标列名称
    :return:
        X: 标准化后的特征矩阵
        y: 目标向量
        feature_names: 特征名称列表
    """

    df = pd.read_csv(file_path)

    # 分离特征和目标
    features = df.drop(columns=[y_column])  # 删去因变量那一列

    # 保存统计量
    feature_means = features.mean().values  # 均值向量
    feature_stds = features.std().values    # 标准差向量
```

```
    feature_names = features.columns.tolist()  # 提取列表名称
    y = df[y_column].values  # 提取因变量值

    # Z-score标准化
    features = (features - features.mean()) / features.std()

    # 在输入矩阵最左侧插入x0(全为1的列)
    X = np.insert(features.values, 0, 1, axis=1)
    feature_names = ['w_0'] + feature_names

    return X, y, feature_names, feature_means, feature_stds


# 定义线性回归模型
def predict(X, w):
    predictions = np.dot(X, w)
    return predictions


# 定义损失函数（MSE）
def loss(X, w, y):
    m = len(y)
    predictions = predict(X, w)
    return (1 / (2 * m)) * np.sum((predictions - y) ** 2)


# 梯度下降法
def gradient_descent(X, y, w_original, alpha, epochs):
    """
    向量化梯度下降
    :param X: 特征矩阵 (m x (n+1))
    :param y: 目标向量 (m x 1)
    :param w_original: 初始权重向量((n+1) x 1)
    :param alpha: 学习率
    :param epochs: 迭代次数
    :return:
        weights: 优化后的权重
        loss_history: 损失记录
    """

    m = len(y)
    w_new = w_original.copy()
    loss_history = []

    for epoch in range(1, epochs + 1):
        # 计算梯度
        predictions = predict(X, w_new)
        error = predictions - y
        gradients = (1 / m) * np.dot(X.T, error)

        # 更新权重
        w_new -= alpha * gradients

        # 记录损失
        current_loss = loss(X, w_new, y)
        loss_history.append(current_loss)
```

```
        # 打印进度
        print(f"Epoch {epoch:4d}/{epochs} | Loss: {current_loss:.4f}")

    return w_new, loss_history


# 将标准化后的权重还原到原始特征空间
def restore_parameters(weights, feature_means, feature_stds):
    w0_prime = weights[0]
    w_prime = weights[1:]

    # 计算原始截距项
    w0 = w0_prime - np.sum((w_prime * feature_means) / feature_stds)

    # 计算原始特征权重
    w = w_prime / feature_stds

    return np.concatenate([[w0], w])


# 主函数
def main():
    file_path = "C:/Users/Administrator/Desktop/pytorchlearning/Folds5x2_pp.csv"
  # ！！！此处替换为自己的数据集路径
    y_column = "PE"  # ！！！查看你的数据集，找到因变量那一列,填写这列的列名

    # 加载数据
    X, y, feature_names, feature_means, feature_stds = load_data(file_path,
y_column)

    # 参数初始化
    np.random.seed()
    w_original = np.random.randn(X.shape[1])
    alpha = 0.01
    epochs = 10000

    # 训练模型
    final_weights, loss_history = gradient_descent(X, y, w_original, alpha,
epochs)

    # 输出标准化下训练的结果
    print("标准化下优化后的权重：")
    for name, weight in zip(feature_names, final_weights):
        print(f"{name}: {weight}")

    # 返回参数
    w = restore_parameters(final_weights, feature_means, feature_stds)

    # 输出标准化下训练的结果
    print("返回后的权重：")
    for name, weight in zip(feature_names, w):
        print(f"{name}: {weight}")

    # 绘制损失曲线
    plt.figure(figsize=(10, 6))
    plt.plot(loss_history, 'r-', linewidth=5)
    plt.title('Training Loss History')
    plt.xlabel('Epoch')
```

```
    plt.ylabel('Loss')
    plt.grid(True)
    plt.show()


if __name__ == "__main__":
    main()
```

6.2 使用说明（一共三步）

1. 将主函数中的file_path替换为自己的数据集路径（csv格式）；
2. 将主函数中的y_column替换为你的数据集因变量列的列名。
3. 然后就可以运行啦~