机器学习第二次作业

朱一凡　计算机学院　2120200467

题目介绍

1.(20 分) 在讲稿第 9 页最下面的“注意”中指出：通过令两个偏导数为零，我们获得了最优参数值的必要条件，它们意味着预测误差是零均值且与输入不相关。试说明“零均值”和“不相关”的根据是什么？

2. 线性回归问题：假设存在直线方程 $y = 1.4x + 0.9$，噪声服从分布 $\mathcal{N}(0, 5)$。

1)(10 分) 生成满足条件的 100 个独立同分布样本，选择其中 80% 作为训练样本，剩余为测试样本。

2)(10 分) 绘出训练样本和测试样本的散布图。

3)(30 分) 编程实现求解线性回归的最小二乘法和采用 Huber 损失函数的鲁棒线性回归方法。

4)(10 分) 使用 3) 中实现的线性回归方法求解训练样本的线性回归方程，并给出测试样本的均方误差。

5)(20 分) 解释 4) 中你所得到的结果，并对两种求解方法进行比较。

1. 第一题

1.1. 问题描述

通过两个偏导数为 0 这一条件推导出此时预测误差是零均值且与输入不相关这一结论。问题难点在于构建出偏导数为 0 和结论之间连接的桥梁。

1.2. 基本思路

输入误差是零均值意味着有下列式子的成立, 其中 y 为观测输出，x 为输入，w 为估计的权值。

$$E(y_i - w_0 - w_1 x_i) = \sum_{i=0}^{n} y_i - w_0 - w_1 x_i = 0 \tag{1}$$

与输入不相关意味着对于任意输入，结论是不变化的。

为了得到上述结论，可以将导数为 0 时的式子进行变形。

1.3. 解题步骤

当两个导数都为 0 时，有以下式子成立：

$$\frac{\partial}{\partial w_0} J_N(\mathrm{w}) = -\sum_{i=1}^{N} (y_i - w_0 - w_1 x_i) = 0 \tag{2}$$

$$\frac{\partial}{\partial w_1} J_N(\mathrm{w}) = -\sum_{i=1}^{N} (y_i - w_0 - w_1 x_i)\, x_i = 0 \tag{3}$$

当式2成立时，有 $\sum_{i=0}^{n}(y_i - w_0 - w_1 x_i) = 0$，所以预测误差的均值为 0。对于式3，有 $(\sum_{i=0}^{n}(y_i - w_0 - w_1 x_i))(\sum_{i=0}^{n} x_i) = 0$，故其与输入 x_i 不相关

2. 第二题

2.1. 问题描述

该问题是一个线性回归问题，需要自己完成数据的生成和对两种回归算法的比较分析。难点在于编程实现时的算法细节处理。

2.2. 基本思路

首先使用随机数和采样实现算法的数据的生成。在生成数据的时候，首先生成 x，在实验中使用 $x \in [-10, 10]$, 均匀的在 x 的区间进行采样。然后根据公式 $y = 1.4x + 0.9 + \mathcal{N}(0, 5)$ 来得到 y 的值。随机误差可使用 numpy 所自带的库来生成。

在得到数据后，编程实现两种回归方法。最小二乘回归比较简单，可以直接使用表达式得到最终的结果。Huber 损失函数的鲁棒线性回归方法则需要使用梯度下降法来进行求解。梯度下降法可以使用 pytorch 进行实现。

2.3. 解题步骤

2.3.1 处理流程

首先进行数据的生成。在算法中使用均匀采样的方法在 $[-10, 10]$ 区间上采样 100 次得到 x, 随后使用公式 $y = 1.4x + 0.9 + \mathcal{N}(0, 5)$ 来得到 y 的值。在得到全部的数据之后，使用随机抽样的方法来确定训练集和测试集。每一次的数据使用一个类函数来表达。绘制散点图可以使用 Python 中的 matplotlib 库进行实现。

在得到训练数据和测试数据之后，可以进行回归算法的实现。实现两种回归算法，分别计算这两种算法在测试集上的均方误差。详细的计算过程在算法描述部分进行讨论。

2.3.2 算法描述

首先进行数据的生成。在算法中使用均匀采样的方法在 $[-10, 10]$ 区间上采样 100 次得到 x, 随后使用公式 $y = 1.4x + 0.9 + \mathcal{N}(0, 5)$ 来得到 y 的值。在得到全部的数据之后，使用随机抽样的方法来确定训练集和测试集。每一次的数据使用一个类函数来表达。绘制散点图可以使用 Python 中的 matplotlib 库进行实现。由于随机数的生成具有不确定性，所以进行多次生成以确保算法的有效性。

在得到训练数据和测试数据之后，可以进行回归算法的实现。对于最小二乘法来说，在处理线性回归时，可以使用以下的公式进行计算

$$\hat{w} = (X^T X)^{-1} X^T y \tag{4}$$

均方误差可以使用以下公式进行计算

$$MSE = \sum_{i=0}^{n} (y - \hat{y})^2 \tag{5}$$

对于 Huber 损失函数的鲁棒线性回归方法，使用 Huber 损失函数作为最终的优化目标。其形式如下：

$$L_{Huber}(r, \delta) = \begin{cases} r^2/2 & \text{if } |r| \le \delta \\ \delta|r| - \delta^2/2 & \text{otherwise} \end{cases} \tag{6}$$

随后使用梯度下降法进行求解。梯度下降法使用 pytorch 实现，在算法中仅这一部分借助库函数实现。

2.3.3 算法实现（Python 语言源代码）

算法的实现结果如附件中的代码所示。使用 Python 语言实现，主要使用了 numpy，matplotlib，pytorch 算法库。

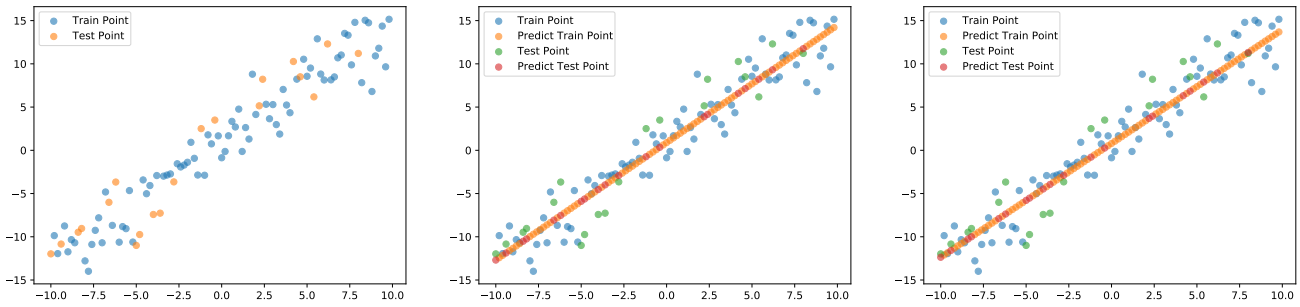
2.4. 结果与分析

2.4.1 实验内容与步骤

实验内容主要由三部分构成，数据的生成，最小二乘算法回归分析，Huber 损失函数法回归分析。首先将数据的范围固定到 $[-10, 10]$，生成 100 组实验数据进行图像的绘制和回归分析。随后改变数据的取值范围，观察其对算法的影响。其中 Huber 的参数为 0.1，最大训练 200 次。

2.4.2 实验结果

首先将数据的范围固定到 $[-10, 10]$，生成 100 组实验数据进行图像的绘制和回归分析. 结果如图1



(a) 数据分布　(b) 最小二乘法结果　(c) Huber 损失函数法结果

图 1. $x \in [-10, 10]$

最终结果：

最小二乘法：$y = 1.35x + 0.88$, $MSE = 7.94$

Huber 法损失函数:$y = 1.37x + 0.89$, $MSE = 7.65$ 将数据的范围固定到 $[-1, 1]$，生成 100 组实验数据进行图像的绘制和回归分析. 结果如图2

最终结果：

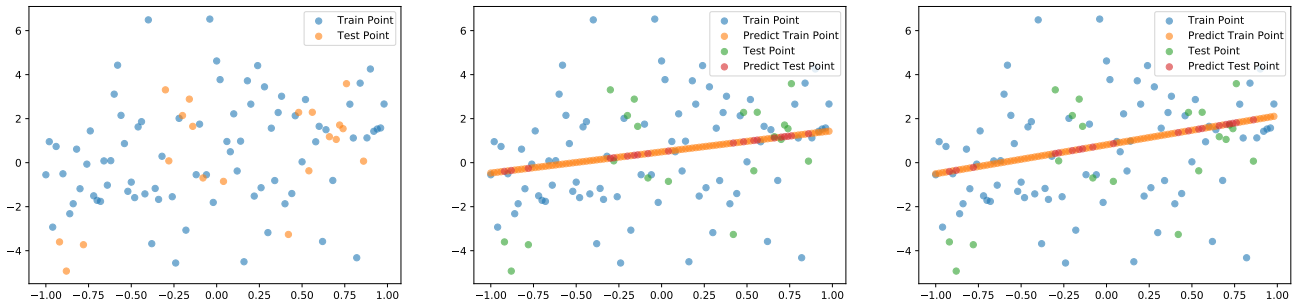
最小二乘法：$y = 0.95x + 0.48$, $MSE = 5.97$

Huber 法损失函数:$y = 1.31x + 0.81$, $MSE = 4.88$ 将数据的范围固定到 $[-100, 100]$，生成 100 组实验数据进行图像的绘制和回归分析. 结果如图3

最终结果：

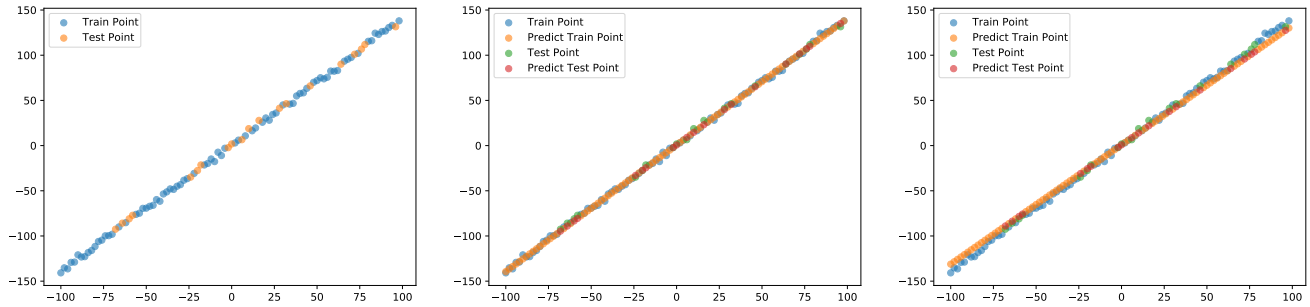
最小二乘法：$y = 1.40x + 0.65$, $MSE = 5.74$

Huber 法损失函数:$y = 1.38x + 0.78$, $MSE = 7.88$



(a) 数据分布　　(b) 最小二乘法结果　　(c) Huber 损失函数法结果

图 2. $x \in [-1, 1]$



(a) 数据分布　　(b) 最小二乘法结果　　(c) Huber 损失函数法结果

图 3. $x \in [-100, 100]$

2.4.3 结果分析

可以看到，在上述的结果中，前两种情形 Huber 方法的均方差都要比最小二乘法要小。这是因为 Huber 损失函数可以很好地减少误差比较大的点对于回归结果的影响。但是最小二乘法却无法做到。但是最小二乘法的计算相比于 Huber 损失函数要更容易计算。在生成数据的区间比较小的时候，生成的数据更难进行回归处理。这是因为当数据范围比较小的时候收到误差的影响要更大，所以更难进行处理。而此时的 Huber 方法相对于最小二乘法的提高也是最大的。但是当数据范围变大时，由于误差点的影响很小，所以此时的最小二乘法有了更好的表现。当然，Huber 的参数也会影响到算法的性能，在此不做讨论。此外，当范围不同的时候，均方误差会因为 y 的值域不同而有所不同。

3. 总结

在这次实验中我体会到了数据分布对于一个算法的成功与否十分的重要。在生成数据范围比较小的时候，很难学习到一个正确的结果。当数据的范围扩大时，才能得到一个比较准确的结果。并且 Huber 损失函数法要比最小二乘法更好地适应离群点对算法的影响，相比于最小二乘法更加稳定。但是不同的算法有不同的使用情况，一定要慎重选择。