机器学习-第五讲课后作业

姓名：周宝航 学号：2120190442 专业：计算机科学与技术

一、问题描述

线性回归问题：数据文件 Dataset(5).csv 中包含 9 个训练样本点，组成训练样本集。

1. 利用之前实现的线性回归最小二乘法得到训练样本集的最小二乘解，并且利用得到的线性回归模型生成 5 个测试样本，组成测试样本集，使用两种颜色绘制出训练样本集和测试样本集中样本的散布图。假设噪声模型为： $N(0,5)$ 。

2. 编程实现岭回归算法，求解出训练样本集的岭回归模型，给出平均训练误差和平均测试误差。

3. 比较岭回归算法与线性回归模型（最小二乘解）的平均训练误差与平均测试误差，给出结论与理由。

4. 如果使用多项式回归模型来拟合训练样本集，应该选择多少阶的多项式？说明你给出此答案的理由。

二、基本思路

1. 我们利用正规方程对训练集求解其最小二乘解，并利用该线性回归模型生成 5 个测试样本。然后，使用 Matplotlib 函数库绘制出训练集和测试集样本的散布图。

2. 我们根据岭回归算法的求解方程，对训练集进行拟合得到回归

模型。考虑到岭回归涉及的超参数，我们采用余一交叉验证进行超参数搜索。然后，固定最优参数得到该模型在训练集和测试集上的平均误差。最后，我们针对线性回归模型（最小二乘解）和岭回归模型进行评价。

3. 多项式回归模型是对一阶线性回归模型的扩展。我们利用余一交叉验证求得最优阶数对应的多项式回归模型。

三、解题步骤

问题 1：

我们利用最小二乘法得到回归模型，然后利用采样数据结合噪声值得到测试集样本。实验代码与运行结果如下：

```
# 加载训练数据
train_data = np.loadtxt('Dataset(5).csv', delimiter=',', skiprows=1)[:, 1:]

# 最小二乘法拟合
lsm = LeastSquareMethod()
lsm.fit(train_data)

# 生成测试样本
n_test_data = 5
test_data = np.random.uniform(0, 2, size=n_test_data).reshape((-1, 1))
noise = np.random.normal(scale=np.sqrt(5), size=n_test_data).reshape((-1, 1))

pred = lsm.predict(test_data)
test_data = np.hstack((test_data, pred + noise))

# 绘制训练样本和测试样本
plt.scatter(train_data[:,0], train_data[:,1], label='train')
plt.scatter(test_data[:,0], test_data[:,1], label='test')
plt.title('Data')
plt.legend()
plt.show()
```

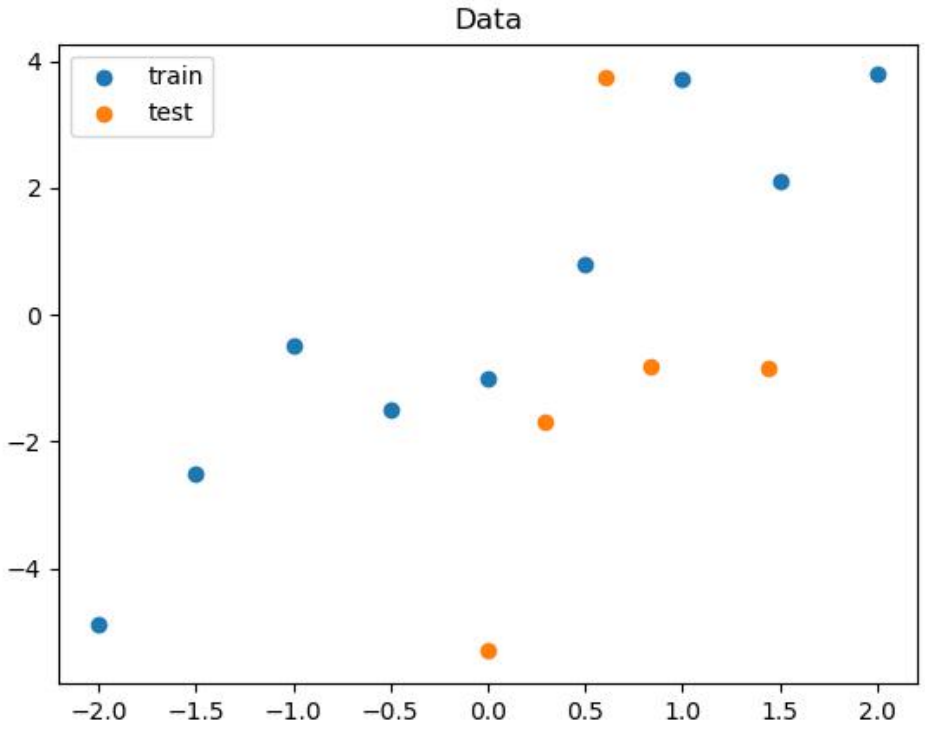

图 1 训练样本与测试样本散布图

问题 2 - 4：

1. 算法描述

1.1 岭回归算法

考虑到线性回归模型的参数值过大会造成输入的变化对输出的影响结果变大。因此，我们引入对参数的限制，重新定义的损失函数为：$J(w)=\frac{1}{n}\sum_{i=1}^{n}(y_i-(w_0+\mathrm{w}^T x_i))^2+\lambda\|w\|_2^2$。其中 λ 称为收缩系数，$\|\mathrm{w}\|_2^2=w^T w$ 是 L2 范数的平方。该方法的最优解为：$\hat{\mathrm{w}}=(\lambda\mathrm{I}+\mathrm{X}^{\mathrm{T}}X)^{-1}X^T y$。该方法通过给模型参数添加高斯先验概率来鼓励其取绝对值较小的数值。

1.2 多项式回归算法

一元多项式回归算法可以表示为：$f(x;w)=w_0+w_1x_1+\ldots+w_m x^m$。然后，我们将参数矩阵与多项式特征值矩阵按照正规方程进行求解参数。

2. 算法实现

2.1 岭回归算法

```
1.  class RidgeRegression(object):
2.      '''''
3.          岭回归算法
4.      '''
5.      def __init__(self, lamb=1.):
6.          self.lamb = lamb
7.          self.params = None
8.
9.      def fit(self, data, verbose=False):
10.         # 样本维度
11.         x = data[:, :-1].reshape((-1, 1))
12.         # 预测值维度
13.         y = data[:, -1].reshape((-1, 1))
14.         # 加入常数项
15.         c_x = np.hstack((np.ones_like(x), x))
16.
17.         # 求解参数
18.         tmp = c_x.T @ c_x
19.         w = np.linalg.inv(self.lamb*np.eye(len(tmp)) + tmp) @ c_x.T @ y
20.         self.params = w
21.         if verbose:
22.             # 训练误差
23.             pred = c_x @ self.params
24.             mse = mean_squared_error(pred, y)
25.             print(f"{self.__class__.__name__}\t 训练误差:{mse}")
26.             plt.plot(x, pred, '-', label='ridge regression')
27.
28.     def predict(self, x):
29.         x = x.reshape((-1, 1))
30.         # 加入常数项
31.         c_x = np.hstack((np.ones_like(x), x))
32.         # 预测结果
33.         pred = c_x @ self.params
34.         return pred
```

上面是我们对岭回归算法的实现类，其中第 19 行是利用求解方程对最优参数进行求解。对于参数λ的选择，我们在后面介绍采用余一交叉验证方法对超参数进行搜索。

2.2 多项式回归算法

```
class PolynomialRegression(object):
    '''''
        一元多项式回归算法
    '''
    def __init__(self, degree=2.):
        self.params = None
        self.degree = degree

    def __create_polynomial(self, x):
        polynomial = np.ones_like(x)

        for i in range(1, self.degree+1):
            polynomial = np.hstack((polynomial, np.power(x, i)))

        return polynomial

    def fit(self, data, verbose=False):
        # 样本维度
        x = data[:, :-1].reshape((-1, 1))
        # 预测值维度
        y = data[:, -1].reshape((-1, 1))
        # 加入常数项
        c_x = self.__create_polynomial(x)

        # 求解正规方程
        w = np.linalg.inv(c_x.T @ c_x) @ c_x.T @ y
        self.params = w

        if verbose:
            # 训练误差
            pred = c_x @ self.params
            mse = mean_squared_error(pred, y)
            print(f"{self.__class__.__name__}\t 训练误差:{mse}")
            plt.plot(x, pred, '-', label='least squares')

    def predict(self, x):
        x = x.reshape((-1, 1))
        # 加入常数项
        c_x = self.__create_polynomial(x)
        # 预测结果
        pred = c_x @ self.params
        return pred
```

我们定义了多项式回归求解类，其中第 26 行是按照正规方程进行参数求解。该方法还涉及一个多项式阶数的选择问题，我们在实验中采用余一交叉验证的方法进行选择。代码如下：

```
1.  def cross_validation(model, data, params={}):
2.      '''''
3.          余一交叉验证
4.      '''
5.      result = {}
6.      param_name = list(params.keys())[0]
7.      for param in params[param_name]:
8.          exec(f"model.{param_name} = param")
9.          errors = []
10.         for i in range(len(data)):
11.             tmp = data.tolist()
12.             # 余一测试样本
13.             test_data = np.array(tmp[i]).reshape((1, -1))
14.             del tmp[i]
15.             # 训练样本
16.             train_data = np.array(tmp)
17.             # 模型训练
18.             model.fit(train_data, verbose=False)
19.             # 模型预测
20.             pred = model.predict(test_data)
21.             # 预测误差
22.             errors.append(np.mean(np.square(pred.flatten() - test_data[:, -1]
    )))
23.         # 每个超参数平均误差
24.         result[param] = np.mean(errors)
25.
26.     min_error = 100.
27.     best_param = 0.
28.     for k, v in result.items():
29.         print(f"参数:{k} 误差:{v}")
30.         if v < min_error:
31.             best_param = k
32.             min_error = v
33.
34.     return best_param, result.values()
```

四、结果与分析

1. 实验内容与步骤

针对岭回归算法，我们需要求解参数方程后，将回归模型用于测试集样本，并采用均方误差作为评价指标衡量模型的优劣。而该算法涉及参数λ的选择。因此，我们需要设置一组待选参数，并利用余一交叉验证法进行最优λ值的选择。然后，固定最优λ值后，将全部训练集用于回归模型的训练。最后，同样将模型应用于测试集，并采用均方误差作为评价指标衡量模型的优劣。均方误差定义如下：

$$\mathrm{MSE}=\frac{1}{m}\sum_{i=1}^{m}(y_i-\hat{y}_i)^2 \text{。}$$

对于多项式回归算法的阶数选择问题，我们采用同上面一样的策略。

2. 实验结果

关于岭回归方法涉及的参数λ选择问题，我们采用余一交叉验证方法进行实验，得到结果如下：

λ取值	e^{-1}	e^{-2}	e^{-3}	e^{-4}	e^{-5}
均方误差	3.3049	3.5439	3.6380	3.6736	3.6868

表 1 参数λ交叉验证结果

根据上面的交叉验证结果，我们在λ取值为：e^{-1}时，取得了交叉验证的最优结果。然后，我们固定该值重新训练岭回归模型并与最小二乘法进行性能比较。

方法	最小二乘法	岭回归算法

训练集误差	0.9258	0.9294
测试集误差	11.8698	11.7477

表 2 模型误差评价

上表是对最小二乘法与岭回归算法性能评价的比较结果。我们可以看到：在采用相同训练集与测试集的情况下，前者的训练误差更小，而后者的测试误差更小。这表明：岭回归算法的鲁棒性更强，面对未知数据的预测能力更准确。

3. 结果分析

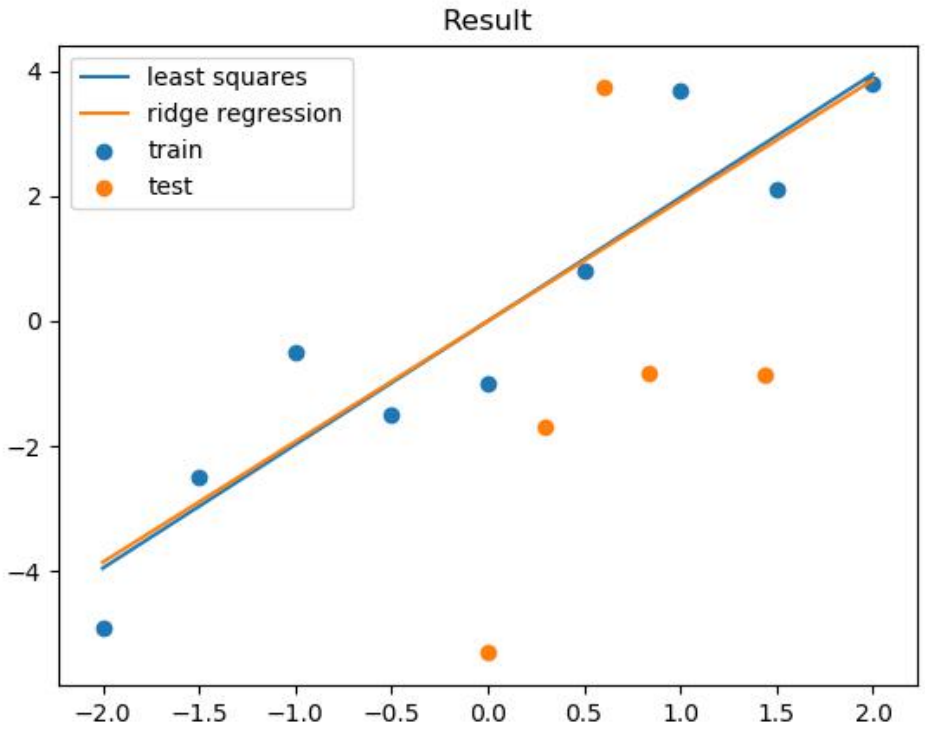

图 2 回归结果图

表 2 展示了最小二乘法和岭回归算法性能评价的比较结果。为了更直观的看到两种方法的结果差异，我们在图 2 绘制了它们各自的回归直线。岭回归算法依靠参数 λ 来控制回归模型参数的绝对值大小。

由图 2 看出，岭回归算法的回归直线较最小二乘法的回归直线的斜率更小，表明：前者的参数学习得到了限制。我们看到：在相同的训练数据情况下，经过模型训练，最小二乘法的测试结果比岭回归方法要差，前者比后者的泛化能力要差。这体现了岭回归算法的优势。

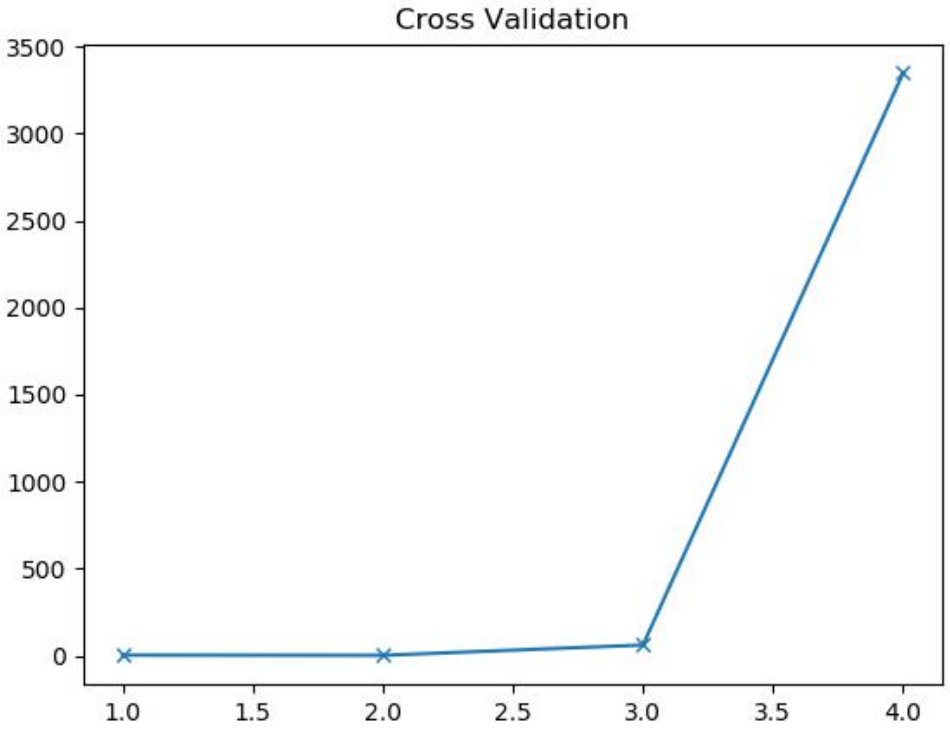

图 3 多项式回归算法交叉验证图

对于多项式回归算法的阶数选择问题，我们采用交叉验证的方式进行解决。我们得到了图 3 所示的结果图，其横坐标为阶数，纵坐标为误差。随着阶数的增加，误差先减少后增加。当阶数为 2 时，多项式回归模型的交叉验证误差取得最小值。因此，在该训练集上，多项式回归模型的阶数选择为：2。