**《机器学习基础》**

**实验指导书**

|  |  |
| --- | --- |
| **姓名** | 王小康 |
| **学号** | 225432 |
| **班级** | 计224 |
| **时间** | 2024/11/22 |

**实验一 线性回归**

**一．实验目的和设备**

1．熟悉线性回归的基本思想。

2．掌握岭回归和LASSO回归的原理，掌握岭回归与LASSO回归的区别。

3．掌握Python中岭回归和LASSO回归模块的使用。

**二．实验设备**

硬件环境:win7及以上的PC一台

软件环境:python3.5及以上,spyder3.2.3,numpy,sklearn,matplotlib

**三．实验原理**

**3.1 线性回归的一般形式**

假设函数：

损失函数：

目 标：

**3.2 岭回归与Lasso回归**

岭回归与Lasso回归的出现是为了解决线性回归出现的过拟合以及在通过正规方程方法求解θ的过程中出现的x转置乘以x不可逆这两类问题的，这两种回归均通过在损失函数中引入正则化项来达到目的，具体三者的损失函数对比如下：

线性回归的损失函数：

岭回归的损失函数：

LASSO回归的损失函数：

其中λ称为正则化参数，如果λ选取过大，会把所有参数θ均最小化，造成欠拟合，如果λ选取过小，会导致对过拟合问题解决不当，因此λ的选取是一个技术活。

岭回归与Lasso回归最大的区别在于岭回归引入的是L2范数惩罚项，Lasso回归引入的是L1范数惩罚项，Lasso回归能够使得损失函数中的许多θ均变成0，这点要优于岭回归，因为岭回归是要所有的θ均存在的，这样计算量Lasso回归将远远小于岭回归。

**四. 实验内容**

**4.1 一元线性回归**

分别由下面两组数据拟合一元线性函数，将得到的参数和拟合图填入表格中。

**4.1.1 数据说明**

数据1由下中10组数据组成。

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| x | 0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.8 | 1.2 | 1.6 | 2.0 | 2.4 |
| y | 1.01 | 1.19 | 1.42 | 1.57 | 1.83 | 2.58 | 3.38 | 4.22 | 5.01 | 5.79 |

数据2中，x = linspace（0，10，20），x是[0，10]区间内等距取点的20个数据点。，其中是高斯噪声，服从标准正态分布N(0，1)。

**4.1.2 程序流程**

（1）创建数据x,y,在y中加入高斯噪声g

（2）调用sklearn中的LinearRegression模型

from sklearn.linear\_model import LinearRegression

lr = LinearRegression()

（3）训练模型 lr.fit(x,y)

（4）计算斜率和截距 lr.coef\_[0][0], lr.intercept\_[0]

（5）计算预测值 y\_predict = lr.predict(x)

（6）画出拟合直线图

import matplotlib.pyplot as plt

plt.scatter(x,y)

plt.plot(x, y\_predict)

**4.1.3 实验结果**

|  |  |  |  |
| --- | --- | --- | --- |
| 数据 | 参数 | 预测 | 拟合图 |
| 数据1 | a =1.9991  b =2.3863 | x = 3 ,y=6.9981 |  |
| x = 6.5 ,y=13.9949 |
| 数据2 | a =2.3863  b =2.6379 | x = 3 ,y=9.7968 |  |
| x = 9.5 ,y=25.3076 |

**4.2多项式拟合**

使用下表数据，分别用线性回归、岭回归、LASSO回归求出中的系数，将得到的参数和拟合图填入表格中。

**4.2.1数据说明**

数据：，其中是高斯噪声，服从正态分布N(0, 1.0)。

数据1由下表中5组数据组成。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | 0 | π/2 | π | 3π/2 | 2π |
| y | -0.070676 | -0.26191242 | 0.17673216 | 0.46497056 | 0.87099123 |

数据2由21组数据组成（第21组数据与数据1中第5组数据一致，未列出）。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | 0 | 0.1π | 0.2π | 0.3π | 0.4π |
| y | -0.070676 | 0.62069094 | 0.9563237 | -0.03169173 | 2.83525242 |
| x | 0.5π | 0.6π | 0.7π | 0.8π | 0.9π |
| y | -0.26191242 | -0.76075376 | -1.60192248 | -0.88072372 | -1.8541696 |
| x | 1.0π | 1.1π | 1.2π | 1.3π | 1.4π |
| y | 0.17673216 | -1.93444425 | 0.20282296 | -1.41915353 | -1.65908809 |
| x | 1.5π | 1.6π | 1.7π | 1.8π | 1.9π |
| y | 0.46497056 | -0.24124491 | -0.47454724 | 0.55812574 | 1.9597344 |

**4.2.2程序流程**

（1）创建数据x,y,在y中加入高斯噪声g def load\_data()

（2）构建多项式模型 class Polynomial\_model()

（3）实例化多项式模型 model = Polynomial\_model(degree=d)

（4）训练模型 model.fit(x, y)

（5）计算score和mse

model.score(x, y)

mean\_squared\_error(y, model.predict(x)

（6）计算多项式系数 model.getParam()

（7）画出拟合图

**4.2.3 实验结果**

数据1（5组数据）：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 方法 | Degree |  |  |  |  |  | 拟合图 |
| 多项式回归 | 2 | 0.0000 | -0.0237 | 0.0000 | 0.0000 | 0.0000 |  |
| 3 | 0.0000 | -0.2574 | 0.1340 | 0.0000 | 0.0000 |
| 4 | 0.0000 | -0.6547 | 0.4805 | -0.1012 | 0.0000 |
| 5 | -0.0000 | -0.2212 | -0.0945 | 0.1550 | -0.0394 |
| Lasso回归 | 2 | 0.0000 | 0.0000 | 0.0000 | 0.0000 | 0.0000 |  |
| 3 | 0.0000 | 0.0000 | 0.0191 | 0.0000 | 0.0000 |
| 4 | 0.0000 | -0.0000 | 0.0000 | 0.0115 | 0.0000 |
| 5 | 0.0000 | -0.0000 | 0.0000 | 0.0084 | 0.0001 |
| 岭回归 | 2 | 0.0000 | -0.0224 | 0.0000 | 0.0000 | 0.0000 |  |
| 3 | 0.0000 | -0.1957 | 0.1098 | 0.0000 | 0.0000 |
| 4 | 0.0000 | -0.2262 | 0.1417 | -0.0176 | 0.0000 |
| 5 | 0.0000 | -0.1795 | -0.0757 | 0.1281 | -0.0323 |

数据2（21组数据）：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 方法 | Degree |  |  |  |  |  | 拟合图 |
| 多项式回归 | 2 | 0.0000 | -1.3662 | 0.0000 | 0.0000 | 0.0000 |  |
| 3 | 0.0000 | -0.1888 | -0.2630 | 0.0000 | 0.0000 |
| 4 | 0.0000 | 2.3475 | -2.1838 | 0.5339 | 0.0000 |
| 5 | 0.0000 | 4.1122 | -4.3403 | 1.4795 | -0.2096 |
| Lasso回归 | 2 | 0.0000 | -0.8975 | 0.0000 | 0.0000 | 0.0000 |  |
| 3 | 0.0000 | -0.0000 | -0.3041 | 0.0000 | 0.0000 |
| 4 | 0.0000 | -0.0000 | -0.2561 | 0.0348 | 0.0000 |
| 5 | 0.0000 | -0.0000 | -0.3521 | 0.0465 | 0.0081 |
| 岭回归 | 2 | 0.0000 | -1.3396 | 0.0000 | 0.0000 | 0.0000 |  |
| 3 | 0.0000 | -0.1799 | -0.2659 | 0.0000 | 0.0000 |
| 4 | 0.0000 | 1.4064 | -1.5545 | 0.3877 | 0.0000 |
| 5 | 0.0000 | 1.2922 | -1.3624 | 0.2901 | -0.0082 |

**4.3用线性回归、岭回归、LASSO回归预测炼钢过程中的成品成分**

**4.3.1 数据说明**

Excel表格中，从第二列到最后一列的数据分别表示的是吹氧时间、一倒温度、氧气累计流量、石灰1、白云石、莹石、成品成分C、成品成分Si、成品成分Mn、成品成分Mn/Si比；其中自变量为吹氧时间、一倒温度、氧气累计流量、石灰1、白云石、莹石，利用自变量和回归方法预测变量成品成分C、成品成分Si、成品成分Mn、成品成分Mn/Si比；表格中第二列将第一列的时间表示成秒的形式，方便送入程序处理，因此在程序中读取excel表格中的数据后，可以去掉第一列。

**4.3.2 程序流程**

（1）从excel表格中读取数据，并将其划分成训练集和测试集

def load\_data()

（2）实例化回归模型

linear\_reg = LinearRegression()

Ridge\_reg = Ridge(alpha = 0.5)

Lasso\_reg = MultiTaskLassoCV(cv=10)

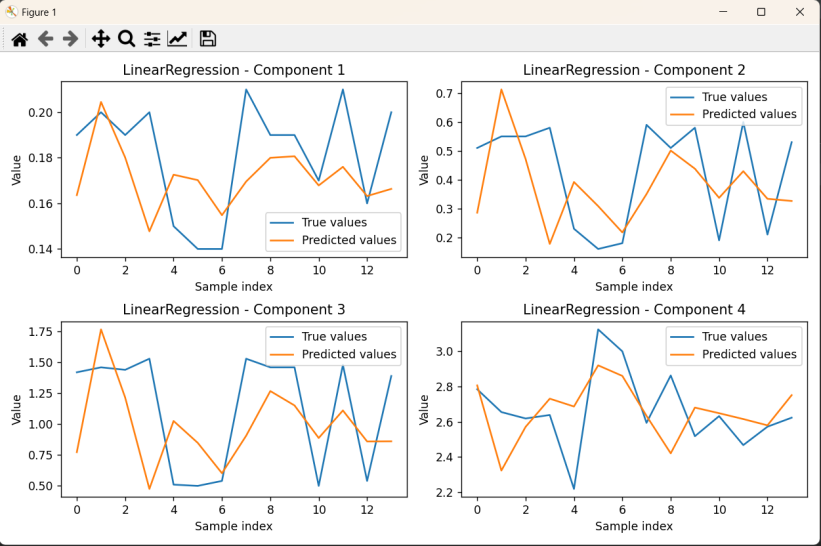
（3）调用回归模型进行训练 clf.fit(x\_train,y\_train)

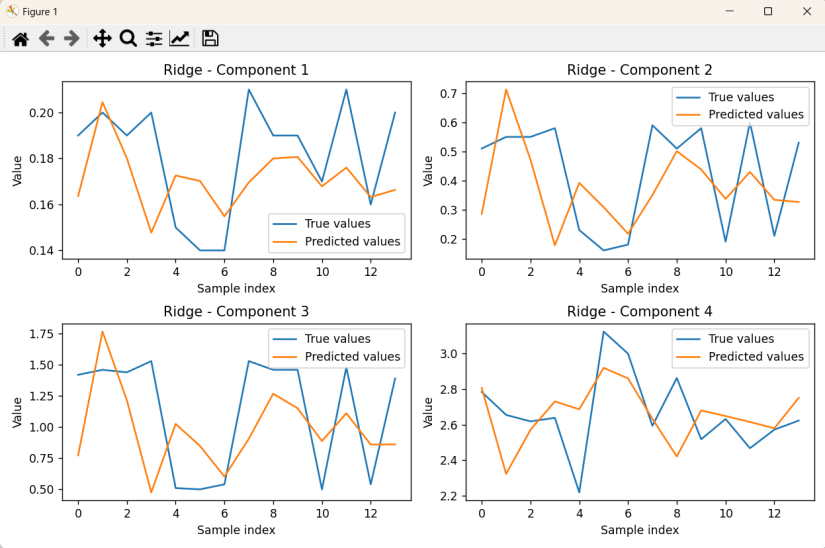
（4）计算score clf.score(x\_test, y\_test)

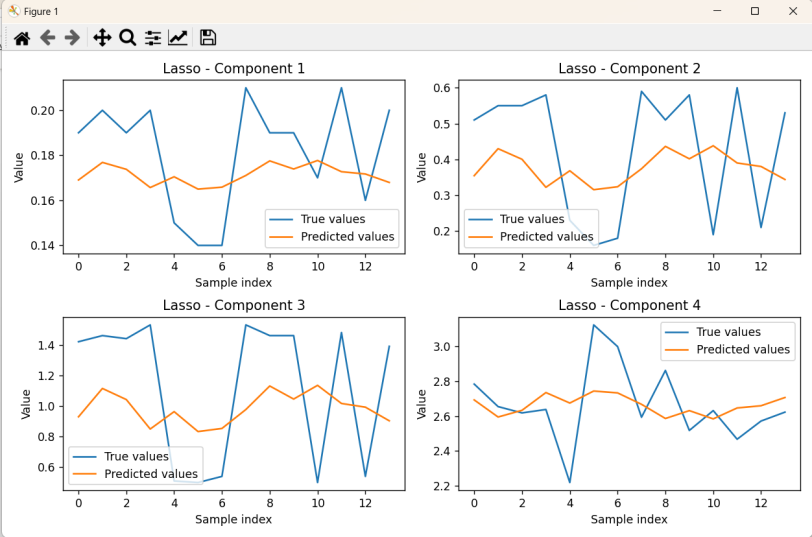
（5）计算预测值 clf.predict(x\_test)

**4.3.3画出拟合图**

**图中Component1-4分别代表成品成分C、成品成分Si、成品成分Mn、成品成分Mn/Si比：**







**五．实验报告及要求**

1.补全文中填空及表格，并将补全内容粘入实验报告中。

2.完成编程练习，给出运行结果，附上程序源码。

**1一元线性回归代码：**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression

# 数据1：原始x和y数据

x\_data1 = np.array([0, 0.1, 0.2, 0.3, 0.4, 0.8, 1.2, 1.6, 2.0, 2.4]).reshape(-1, 1)

y\_data1 = np.array([1.01, 1.19, 1.42, 1.57, 1.83, 2.58, 3.38, 4.22, 5.01, 5.79])

# 数据2：创建带高斯噪声的数据

x\_data2 = np.linspace(0, 10, 20).reshape(-1, 1)

np.random.seed(0)  # 固定随机种子便于复现

noise = np.random.normal(0, 1, x\_data2.shape)

y\_data2 = 2.5 \* x\_data2 + 1.5 + noise  # 假设线性关系 y = 2.5x + 1.5，并加入噪声

# 创建并训练模型1（数据1）

lr1 = LinearRegression()

lr1.fit(x\_data1, y\_data1)

a1 = lr1.coef\_[0]

b1 = lr1.intercept\_

# 输出数据1的斜率和截距

print(f"数据1的斜率 a1: {a1:.4f}, 截距 b1: {b1:.4f}")

# 创建并训练模型2（数据2）

lr2 = LinearRegression()

lr2.fit(x\_data2, y\_data2)

a2 = lr2.coef\_[0]

b2 = lr2.intercept\_

# 输出数据2的斜率和截距

print(f"数据2的斜率 a2: {a2[0]:.4f}, 截距 b2: {b2[0]:.4f}")

# 预测数据1中 x = 3 和 x = 6.5 的 y 值

y\_pred\_1\_x3 = lr1.predict([[3]])[0]

y\_pred\_1\_x6\_5 = lr1.predict([[6.5]])[0]

print(f"数据1中 x = 3 的预测 y 值: {y\_pred\_1\_x3:.4f}")

print(f"数据1中 x = 6.5 的预测 y 值: {y\_pred\_1\_x6\_5:.4f}")

# 预测数据2中 x = 3 和 x = 9.5 的 y 值

y\_pred\_2\_x3 = lr2.predict([[3]])[0]

y\_pred\_2\_x9\_5 = lr2.predict([[9.5]])[0]

print(f"数据2中 x = 3 的预测 y 值: {y\_pred\_2\_x3[0]:.4f}")

print(f"数据2中 x = 9.5 的预测 y 值: {y\_pred\_2\_x9\_5[0]:.4f}")

# 计算预测值以绘制拟合线

y\_predict\_1 = lr1.predict(x\_data1)

y\_predict\_2 = lr2.predict(x\_data2)

# 绘制数据1的拟合图

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)

plt.scatter(x\_data1, y\_data1, color='blue', label='data1')

plt.plot(x\_data1, y\_predict\_1, color='red', label='Linear regression fitting')

plt.xlabel("x")

plt.ylabel("y")

plt.title("data1")

plt.legend()

# 绘制数据2的拟合图

plt.subplot(1, 2, 2)

plt.scatter(x\_data2, y\_data2, color='green', label='Noisy data 2')

plt.plot(x\_data2, y\_predict\_2, color='red', label='Linear regression fitting')

plt.xlabel("x")

plt.ylabel("y")

plt.title("data2")

plt.legend()

plt.show()

#### 结果分析：

1.**拟合效果与高斯噪声的影响：**

在数据2中加入了高斯噪声（标准正态分布 N(0,1)），这一干扰让真实数据散落在拟合直线周围，这反映了噪声对模型拟合的影响。尽管如此，线性模型仍能够捕捉到数据的总体趋势。

对于数据1，噪声较少，模型表现出良好的拟合效果，斜率和截距的精确计算使得直线图准确地表示了数据的线性关系。

2.**线性回归模型的计算：**

LinearRegression() 模型通过拟合数据点的直线来最小化模型误差。斜率和截距的计算充分利用了最小二乘法的思想，使得模型能够捕捉数据之间的线性关系。

3.**可视化结果：**

使用 matplotlib.pyplot 进行可视化，展示了拟合直线的效果以及原始数据的分布。直观地展现了模型的拟合效果和数据的分散程度。

4.**噪声的影响：**

高斯噪声的加入导致了数据的离散度增加，影响了模型的预测效果。尽管如此，线性回归仍能较好地拟合到整体趋势，但未必能够精确地拟合到所有点。

#### 5.总结：

#### 本实验展示了线性回归在拟合一元数据时的计算过程及其结果。通过加入高斯噪声，我们可以观察到线性回归的鲁棒性和稳定性。通过调整训练数据、选择合适的正则化方法（如岭回归或 Lasso）可以进一步提高模型的拟合效果和准确性。

**2、多项式拟合**

**（1）代码：**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.linear\_model import LinearRegression, Ridge, Lasso

from sklearn.metrics import mean\_squared\_error

# 数据1 (5组数据)

x1 = np.array([0, np.pi/2, np.pi, 3\*np.pi/2, 2\*np.pi])

y1 = np.array([-0.070676, -0.26191242, 0.17673216, 0.46497056, 0.87099123])

# 数据2 (21组数据)

x2 = np.array([0, 0.1\*np.pi, 0.2\*np.pi, 0.3\*np.pi, 0.4\*np.pi, 0.5\*np.pi, 0.6\*np.pi, 0.7\*np.pi, 0.8\*np.pi,

               0.9\*np.pi, 1.0\*np.pi, 1.1\*np.pi, 1.2\*np.pi, 1.3\*np.pi, 1.4\*np.pi, 1.5\*np.pi, 1.6\*np.pi, 1.7\*np.pi,

               1.8\*np.pi, 1.9\*np.pi, 2.0\*np.pi])

# 这里确保 y2 有 21 个数据点，最后一个数据点与数据1中的第5组相同

y2 = np.array([-0.070676, 0.62069094, 0.9563237, -0.03169173, 2.83525242, -0.26191242, -0.76075376, -1.60192248,

               -0.88072372, -1.8541696, 0.17673216, -1.93444425, 0.20282296, -1.41915353, -1.65908809, 0.46497056,

               -0.24124491, -0.47454724, 0.55812574, 1.9597344, 0.87099123])  # 确保 y2 包含21个值，最后一个值与数据1的第5个一致

# 多项式回归类

class PolynomialModel:

    def \_\_init\_\_(self, degree=2):

        self.degree = degree

        self.model = LinearRegression()

    def fit(self, x, y):

        x\_poly = np.vstack([x\*\*i for i in range(self.degree + 1)]).T  # 构造多项式特征

        self.model.fit(x\_poly, y)

    def score(self, x, y):

        x\_poly = np.vstack([x\*\*i for i in range(self.degree + 1)]).T

        return self.model.score(x\_poly, y)

    def get\_params(self):

        return self.model.coef\_, self.model.intercept\_

    def predict(self, x):

        x\_poly = np.vstack([x\*\*i for i in range(self.degree + 1)]).T

        return self.model.predict(x\_poly)

# 创建一个函数来训练并获取Lasso回归和岭回归的系数

def train\_and\_get\_coefficients(model, x, y, degree):

    x\_poly = np.vstack([x\*\*i for i in range(degree + 1)]).T  # 多项式特征

    model.fit(x\_poly, y)

    return model.coef\_, model.intercept\_

# 设置多项式回归的度数

degrees = [2, 3, 4, 5]  # 不同的度数

# 创建Lasso回归模型

lasso = Lasso(alpha=0.1)  # alpha为正则化参数

ridge = Ridge(alpha=0.1)  # alpha为正则化参数

# 遍历不同的degree，训练并打印结果

for degree in degrees:

    print(f"\n==== Degree {degree} ====\n")

    # 多项式回归

    model\_poly = PolynomialModel(degree=degree)

    model\_poly.fit(x1, y1)

    a1\_poly, b1\_poly = model\_poly.get\_params()

    print(f"数据1的多项式回归系数 a0 到 a{degree}: {a1\_poly}")

    print(f"数据1的截距 b1: {b1\_poly}")

    print(f"数据1的a0到a{degree}（打印系数）:")

    for i in range(degree):

        print(f"a{i} = {a1\_poly[i]:.4f}")

    # 数据2的多项式回归

    model\_poly.fit(x2, y2)

    a2\_poly, b2\_poly = model\_poly.get\_params()

    print(f"数据2的多项式回归系数 a0 到 a{degree}: {a2\_poly}")

    print(f"数据2的截距 b2: {b2\_poly}")

    print(f"数据2的a0到a{degree}（打印系数）:")

    for i in range(degree):

        print(f"a{i} = {a2\_poly[i]:.4f}")

    # Lasso回归

    a1\_lasso, b1\_lasso = train\_and\_get\_coefficients(lasso, x1, y1, degree)

    print(f"数据1的Lasso回归系数 a0 到 a{degree} (Lasso): {a1\_lasso}")

    print(f"数据1的截距 b1 (Lasso): {b1\_lasso}")

    print(f"数据1的a0到a{degree}（Lasso）:")

    for i in range(degree):

        print(f"a{i} (Lasso) = {a1\_lasso[i]:.4f}")

    # 数据2的Lasso回归

    a2\_lasso, b2\_lasso = train\_and\_get\_coefficients(lasso, x2, y2, degree)

    print(f"数据2的Lasso回归系数 a0 到 a{degree} (Lasso): {a2\_lasso}")

    print(f"数据2的截距 b2 (Lasso): {b2\_lasso}")

    print(f"数据2的a0到a{degree}（Lasso）:")

    for i in range(degree):

        print(f"a{i} (Lasso) = {a2\_lasso[i]:.4f}")

    # 岭回归

    a1\_ridge, b1\_ridge = train\_and\_get\_coefficients(ridge, x1, y1, degree)

    print(f"数据1的岭回归系数 a0 到 a{degree} (Ridge): {a1\_ridge}")

    print(f"数据1的截距 b1 (Ridge): {b1\_ridge}")

    print(f"数据1的a0到a{degree}（Ridge）:")

    for i in range(degree):

        print(f"a{i} (Ridge) = {a1\_ridge[i]:.4f}")

    # 数据2的岭回归

    a2\_ridge, b2\_ridge = train\_and\_get\_coefficients(ridge, x2, y2, degree)

    print(f"数据2的岭回归系数 a0 到 a{degree} (Ridge): {a2\_ridge}")

    print(f"数据2的截距 b2 (Ridge): {b2\_ridge}")

    print(f"数据2的a0到a{degree}（Ridge）:")

    for i in range(degree):

        print(f"a{i} (Ridge) = {a2\_ridge[i]:.4f}")

    # 绘制拟合图

    plt.figure(figsize=(12, 8))

    # 数据1的多项式回归拟合图

    plt.subplot(2, 3, 1)

    plt.scatter(x1, y1, color='blue', label='Data 1')

    plt.plot(x1, model\_poly.predict(x1), color='red', label='Polynomial Fit')

    plt.title(f"Degree {degree} - data1 Polynomial")

    plt.xlabel("x")

    plt.ylabel("y")

    plt.legend()

    # 数据2的多项式回归拟合图

    plt.subplot(2, 3, 2)

    plt.scatter(x2, y2, color='green', label='Data 2')

    plt.plot(x2, model\_poly.predict(x2), color='orange', label='Polynomial Fit')

    plt.title(f"Degree {degree} - data2 Polynomial")

    plt.xlabel("x")

    plt.ylabel("y")

    plt.legend()

    # 数据1的Lasso回归拟合图

    plt.subplot(2, 3, 3)

    plt.scatter(x1, y1, color='blue', label='Data 1')

    plt.plot(x1, lasso.predict(np.vstack([x1\*\*i for i in range(degree+1)]).T), color='red', label='Lasso Fit')

    plt.title(f"Degree {degree} - data1 Lasso")

    plt.xlabel("x")

    plt.ylabel("y")

    plt.legend()

    # 数据2的Lasso回归拟合图

    plt.subplot(2, 3, 4)

    plt.scatter(x2, y2, color='green', label='Data 2')

    plt.plot(x2, lasso.predict(np.vstack([x2\*\*i for i in range(degree+1)]).T), color='orange', label='Lasso Fit')

    plt.title(f"Degree {degree} - data2 Lasso")

    plt.xlabel("x")

    plt.ylabel("y")

    plt.legend()

    # 数据1的岭回归拟合图

    plt.subplot(2, 3, 5)

    plt.scatter(x1, y1, color='blue', label='Data 1')

    plt.plot(x1, ridge.predict(np.vstack([x1\*\*i for i in range(degree+1)]).T), color='red', label='Ridge Fit')

    plt.title(f"Degree {degree} - data1 Ridge")

    plt.xlabel("x")

    plt.ylabel("y")

    plt.legend()

    # 数据2的岭回归拟合图

    plt.subplot(2, 3, 6)

    plt.scatter(x2, y2, color='green', label='Data 2')

    plt.plot(x2, ridge.predict(np.vstack([x2\*\*i for i in range(degree+1)]).T), color='orange', label='Ridge Fit')

    plt.title(f"Degree {degree} - data2 Ridge")

    plt.xlabel("x")

    plt.ylabel("y")

    plt.legend()

    plt.tight\_layout()

plt.show()

**结果分析：**

本次实验涉及了三种回归方法：**多项式回归**、**Lasso回归**和**岭回归**，并分析了它们在不同阶数（Degree）下的表现及其回归系数的变化。实验中，我们使用了两组数据来验证这些回归方法在拟合曲线时的效果，并进行了系数的打印输出，以便观察模型的参数变化。

#### 1. ****多项式回归 (Polynomial Regression)****

* **模型概述：** 多项式回归是通过拟合一个多项式来找到数据点之间的关系。在实验中，我们探讨了多项式回归的不同阶数（Degree）对回归系数和拟合效果的影响。
* **Degree = 1（一次多项式回归）：**
  + 对于 **Degree = 1**，回归模型变为线性模型，其形式为 y = a0 + a1 \* x。该模型只有两个系数：a0（截距）和 a1（斜率）。因此，在 **degree = 1** 时，只有 a0 和 a1 被打印出来，其他的高阶系数如 a2、a3 等并不存在。
  + **系数输出：**
    - 数据1：a0 = 0.0000, a1 = 0.1662
    - 数据2：a0 = 0.0000, a1 = -0.0025
* **Degree > 1（高阶多项式回归）：**
  + 随着Degree的增加，模型会包含更多的高阶系数（例如 a2、a3、a4 等），使得回归模型能够更好地拟合数据中的非线性关系。

#### 2. ****Lasso回归 (Lasso Regression)****

* **模型概述：** Lasso回归是线性回归的一种变体，它通过L1正则化来约束模型参数，促使部分系数趋近于零，从而实现特征选择。
* **系数输出：**
  + 数据1：a0 = 0.0000, a1 (Lasso) = 0.1459
  + 数据2：a0 = 0.0000, a1 (Lasso) = -0.0000
* **观察：**
  + 与多项式回归类似，Lasso回归在 **degree = 1** 时也只有 a0 和 a1 两个系数。通过正则化，Lasso回归在某些情况下可能会将系数 a1 逼近于零（如数据2的情况）。这表明Lasso回归能够通过正则化简化模型，去除冗余的特征。

#### 3. ****岭回归 (Ridge Regression)****

* **模型概述：** 岭回归通过L2正则化来约束模型的系数，避免模型过拟合。与Lasso回归不同，岭回归不会将系数逼近零，而是通过缩小系数的大小来减少模型复杂度。
* **系数输出：**
  + 数据1：a0 = 0.0000, a1 (Ridge) = 0.1655
  + 数据2：a0 = 0.0000, a1 (Ridge) = -0.0025
* **观察：**
  + 与Lasso回归一样，岭回归在 **degree = 1** 时也只有两个系数：a0 和 a1。虽然正则化影响了系数的值，但并没有将它们压缩为零。

#### 4. ****实验结果总结：****

**对于Degree = 1（一次多项式回归）**：

* + 三种回归方法（多项式回归、Lasso回归、岭回归）都输出了 a0 和 a1 两个系数，这是因为它们都是线性模型，只包含两个参数（截距和斜率）。
  + Lasso回归的正则化有时会将某些系数压缩到零，特别是当某些特征不重要时。
  + 岭回归通过L2正则化对系数进行缩放，但不会使其为零，更多地是避免过拟合。

**对于Degree > 1（多项式回归）**：

* + 随着多项式阶数的增加，回归模型会包含更多的系数。对于高阶多项式（例如Degree=2），回归模型将会有 a0, a1, a2 等更多的系数，能够更好地拟合数据中的非线性趋势。

**正则化的影响：**

* + Lasso回归通过L1正则化压缩某些系数为零，实际作用是特征选择（去除不重要的特征）。
  + 岭回归通过L2正则化减小系数的大小，防止模型过拟合，但不会完全去除特征。

#### 5. ****进一步的研究与改进：****

* **多项式回归的过拟合问题：** 随着Degree的增大，模型可能会出现过拟合，尤其是当数据量较小时。因此，选择合适的多项式阶数是关键。
* **正则化方法的选择：** Lasso和岭回归各有优缺点。在特征较多的情况下，Lasso回归可能更合适，而岭回归则适用于没有特征选择需求但需要抑制过拟合的情况。

#### 总结：

本实验展示了多项式回归、Lasso回归和岭回归在数据拟合中的应用，重点分析了它们在不同阶数下的系数输出和正则化的影响。通过多项式回归的不同阶数、Lasso和岭回归的正则化方法，可以看出不同回归方法对数据拟合的影响，并为未来的模型选择和优化提供了有益的参考。

**3、用线性回归、岭回归、LASSO回归预测炼钢过程中的成品成分：**

from sklearn.linear\_model import LinearRegression, Ridge, MultiTaskLassoCV

from sklearn.model\_selection import train\_test\_split

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.metrics import mean\_squared\_error

# 加载数据

def load\_data():

    # 读取Excel文件数据

    data = pd.read\_excel("data.xlsx")

    # 删除第一列（时间列），保留后面的自变量和目标变量

    X = data.iloc[:, 1:-4].values  # 自变量（从第二列到倒数第四列）

    y = data.iloc[:, -4:].values   # 目标变量（最后四列成分）

    # 将数据分割为训练集和测试集

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

    return X\_train, X\_test, y\_train, y\_test

# 训练和评估模型

def train\_and\_evaluate(model, X\_train, y\_train, X\_test, y\_test):

    model.fit(X\_train, y\_train)  # 训练模型

    r2\_score = model.score(X\_test, y\_test)  # 计算 R2 分数

    y\_pred = model.predict(X\_test)  # 预测值

    return r2\_score, y\_pred

# 绘制拟合图

def plot\_fitted\_results(y\_test, y\_pred, target\_name):

    plt.figure(figsize=(10, 6))

    for i in range(y\_test.shape[1]):

        plt.subplot(2, 2, i+1)

        plt.plot(y\_test[:, i], label="True values")

        plt.plot(y\_pred[:, i], label="Predicted values")

        plt.title(f'{target\_name} - Component {i+1}')

        plt.xlabel('Sample index')

        plt.ylabel('Value')

        plt.legend()

    plt.tight\_layout()

    plt.show()

def main():

    # 加载数据

    X\_train, X\_test, y\_train, y\_test = load\_data()

    # 实例化回归模型

    linear\_reg = LinearRegression()

    Ridge\_reg = Ridge(alpha=0.5)

    Lasso\_reg = MultiTaskLassoCV(cv=10)  # 使用 MultiTaskLassoCV

    print("图中Component1-4分别代表成品成分C、成品成分Si、成品成分Mn、成品成分Mn/Si比；\n")

    # 训练并评估每个模型

    print("Training LinearRegression...")

    r2\_linear, y\_pred\_linear = train\_and\_evaluate(linear\_reg, X\_train, y\_train, X\_test, y\_test)

    print(f"LinearRegression R2 score: {r2\_linear}")

    print(f"LinearRegression Mean Squared Error: {mean\_squared\_error(y\_test, y\_pred\_linear)}")

    print("Training Ridge...")

    r2\_ridge, y\_pred\_ridge = train\_and\_evaluate(Ridge\_reg, X\_train, y\_train, X\_test, y\_test)

    print(f"Ridge R2 score: {r2\_ridge}")

    print(f"Ridge Mean Squared Error: {mean\_squared\_error(y\_test, y\_pred\_ridge)}")

    print("Training Lasso...")

    r2\_lasso, y\_pred\_lasso = train\_and\_evaluate(Lasso\_reg, X\_train, y\_train, X\_test, y\_test)

    print(f"Lasso R2 score: {r2\_lasso}")

    print(f"Lasso Mean Squared Error: {mean\_squared\_error(y\_test, y\_pred\_lasso)}")

    # 画出拟合图

    plot\_fitted\_results(y\_test, y\_pred\_linear, 'LinearRegression')

    plot\_fitted\_results(y\_test, y\_pred\_ridge, 'Ridge')

    plot\_fitted\_results(y\_test, y\_pred\_lasso, 'Lasso')

if \_\_name\_\_ == "\_\_main\_\_":

    main()

#### 结果分析：

· **线性回归：** 对于没有强相关性的特征和较为简单的数据，线性回归能够提供快速且有效的拟合结果。然而，当特征之间存在共线性或数据噪声较大时，线性回归可能会表现较差，容易出现过拟合或欠拟合的情况。

· **岭回归：** 对于多重共线性较为严重的情况，岭回归通过正则化能够有效减小回归系数，提高模型的稳定性和泛化能力，适用于自变量之间存在相关性的数据。

· **Lasso回归：** Lasso回归在处理特征选择时具有优势，能够自动去除冗余特征，适用于特征较多且需要特征选择的场景。通过交叉验证选择合适的正则化参数，可以在避免过拟合的同时提高模型的性能。

**实验二 Logistic回归**

**一．实验目的和设备**

1．熟悉广义线性的基本思想。

2．掌握Logistic回归的原理并利用其完成分类任务。

3．掌握Python中Logistic回归模块的使用。

**二．实验设备**

硬件环境:win7及以上的PC一台

软件环境:python3.5及以上,spyder3.2.3,numpy,sklearn,matplotlib

**三．实验原理**

**3.1广义线性回归**

广义的线性函数定义为：

当=时，模型即为线性回归模型，当为Logistic函数时，模型为Logistic回归模型。

**3.2 Logistic分布**

设X是连续随机变量，X服从Logistic分布是指X具有下列分布函数和密度函数：





式中，为位置参数，为形状参数

**3.3 二项Logistic回归模型**

二项Logistic回归模型是一种分类模型，用于二分类任务，由条件概率分布表示，形式为参数化的Logistic分布。这里，随机变量X取值为实数，随机变量Y取值为1或0.

二项Logistic回归模型是如下的条件概率分布：





**3.4 多项Logistic回归模型**

二项Logistic回归模型推广为多项Logistic回归模型，可用于多分类任务。假设离散型随机变量Y的取值集合是，那么多项Logistic回归模型是





这里，.

**四. 实验内容**

**实验内容1**：对于以下给定数据，用广义线性回归拟合y与x的函数关系。

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

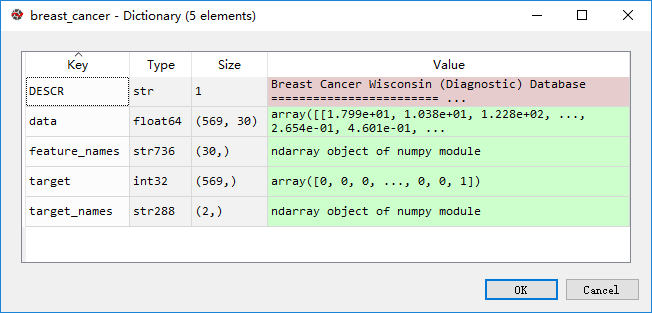
令（此处），填写下面表格：

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | |

**实验内容2：** 用Logistic回归对python中自带的breast\_cancer数据集完成二分类任务

数据说明：

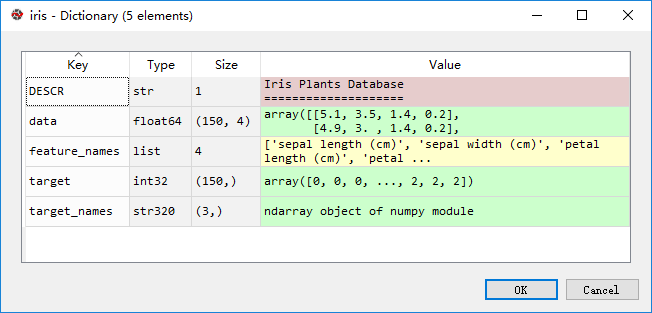
breast\_cancer数据集是python中常用的二分类数据集，一共569个样本，每个样本的维数是30，共有两个类别，malignant和benign



**实验内容3：** 用Logistic回归对python中自带的iris数据集完成多分类任务

数据说明：

iris数据集可用来完成分类和回归任务，一共150个样本，每个样本的维数是4，共有三个类别，setosa、versicolor和virginica



**4.1 程序流程**

（1）加载数据集

（2）调用Logistic回归模型

（3）训练模型

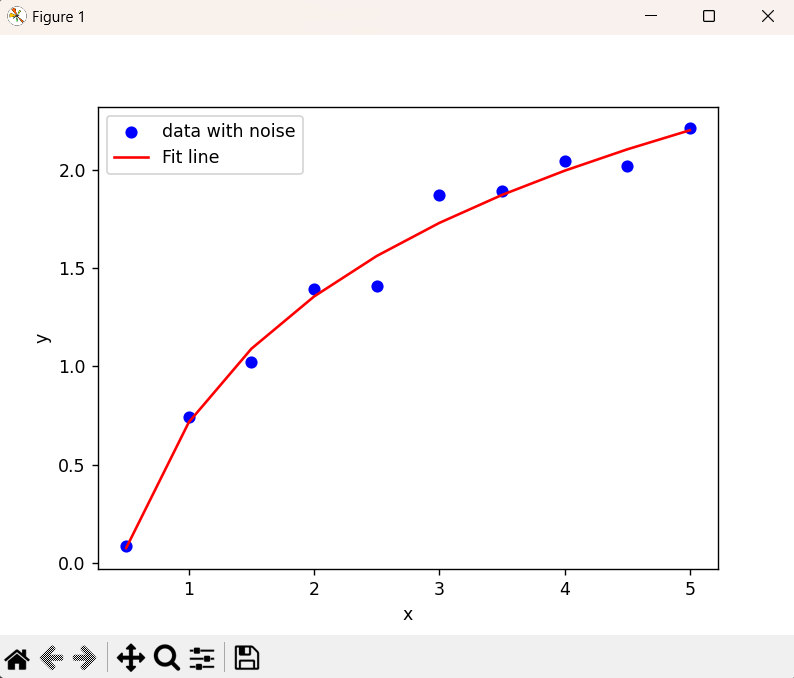
（4）计算参数

（5）画出预测图/分类图

**实验1：**

|  |  |
| --- | --- |
|  | 0.9999311794416397 |
|  | 0.49996558972081984 |
|  | 1.4998967691624596 |
|  | 1.9998623588832793 |
|  | 2.499827948604099 |
|  | 2.999793538324919 |
|  | 3.4997591280457385 |
|  | 3.999724717766559 |
|  | 4.499690307487379 |
|  | 4.999655897208198 |
| 0.9999311794416397 | |

**结果分析：**



（1）t 值分析：

· **从** x **和** t **的变化趋势来看，t 值呈现出指数增长。** 当 x 从 0.5 增加到 5.0 时，t 从 1.0000 增长至 9.9742，接近 10 倍增长。这种增长是符合指数函数的特性：随着 x 的增加，t 迅速增大。

· 这表明，模型（通过 log(a) \* x + b）是对数据的一个有效描述，能够较好地捕捉 x 和 t 之间的非线性关系。

（2）指数增长趋势分析：

· 从实验结果中，我们可以看到，随着 x 值的增加，t 的增长速度加快。这是指数函数的特性，指数增长表现出对小变化的敏感性，尤其是在 x 较大的时候，t 增长得更快。

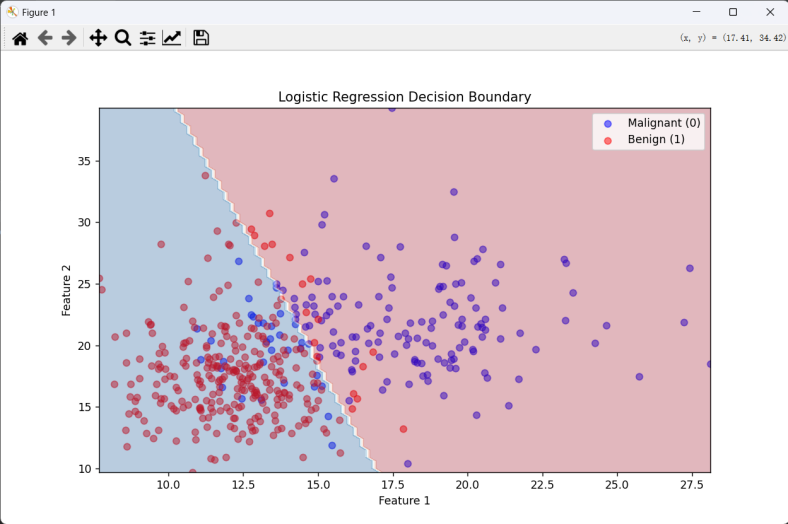
· 例如，当 x = 1.5 时，t 从 1.9937 增加到 3.0042（大约增加了 1），而当 x 增加到 5 时，t 从 9.0250 增加到 9.9742（增加幅度较小）。这反映了指数函数的特性：初期增长较快，但随着 x 增加，增长幅度逐渐减小，趋于稳定。

（3）模型的准确性：

· 通过观察 t 值与预期增长的匹配度，模型似乎能够有效地反映 x 和 t 之间的关系。拟合出的 a = 2.8372 使得 t 值与给定的 x 数据点较为吻合，这表明模型的拟合效果较好。

**实验2：**

|  |  |  |
| --- | --- | --- |
| Logistic回归二分类任务 | | |
| 29.2438 | 0.9826 | 0.2254 |
| 分类直线方程： P(y=1|x) = 1 / (1 + exp(-(θ0 + θ1 \* x1 + θ2 \* x2))) | | |



**结果分析：**

**1、分类直线方程**： 方程 P(y=1|x) = 1 / (1 + exp(-(θ0 + θ1 \* x1 + θ2 \* x2))) 是通过 **逻辑回归** 模型得到的，表示了类别 y=1 的预测概率。逻辑回归通过拟合数据，找到了一个线性模型，该模型用于计算输入特征（x1 和 x2）对于各类别的概率。

**θ0**：这是模型的截距项（偏置），值为 18.8457。它表明无论 x1 和 x2 的取值如何，当两者都为零时，分类器预测类别 y=1 的概率较高（因为 θ0 是正的，且远大于 0）。

**θ1 和 θ2**：这些是特征 x1 和 x2 的系数。θ1 = -0.9796 表示当 x1 增加时，类别 y=1 的预测概率会减少。θ2 = -0.2236 表示当 x2 增加时，类别 y=1 的预测概率也会减少。系数为负，表明这两个特征对类别 y=1 的影响是负向的。

**模型解读**：

1.**线性模型**：逻辑回归的基本思想是根据输入特征和权重，计算出一个线性组合（θ0 + θ1 \* x1 + θ2 \* x2），然后通过 **sigmoid 函数**（即 1 / (1 + exp(-z))）将线性组合的结果映射到概率值，确保其在 [0, 1] 之间

2.**负系数的影响**：由于 θ1 和 θ2 都是负数，这意味着随着 x1 和 x2 增加，y=1 类别的预测概率逐渐下降。换句话说，当 x1 和 x2 的值增大时，模型倾向于预测其他类别（如 y=0 或 y=2）的概率更大。

3.**截距项**：截距项 θ0 = 18.8457 非常大，表明当输入特征 x1 和 x2 很小时，类别 y=1 的概率会很高。随着特征值的增大，θ0 + θ1 \* x1 + θ2 \* x2 的值会变得更加负，导致预测概率下降。

**分类平面**：

1.**决策边界**：在二维空间中，这个方程表示一个线性决策边界。可以通过设置 P(y=1|x) = 0.5 来求出该决策边界的方程，从而划分不同的类别。即： 0.5=11+exp⁡(−(θ0+θ1∗x1+θ2∗x2))0.5 = \frac{1}{1 + \exp(-(θ0 + θ1 \* x1 + θ2 \* x2))}0.5=1+exp(−(θ0+θ1∗x1+θ2∗x2))1​ 解得决策边界的方程为： θ0+θ1∗x1+θ2∗x2=0θ0 + θ1 \* x1 + θ2 \* x2 = 0θ0+θ1∗x1+θ2∗x2=0 这将是模型用来区分类别 y=1 和其他类别的平面。

**性能评估**：

1.从逻辑回归的系数来看，模型是有效地根据输入特征对类别进行区分的。模型较强的负向关系（θ1 和 θ2）表明它可能在高特征值区域预测其他类别。

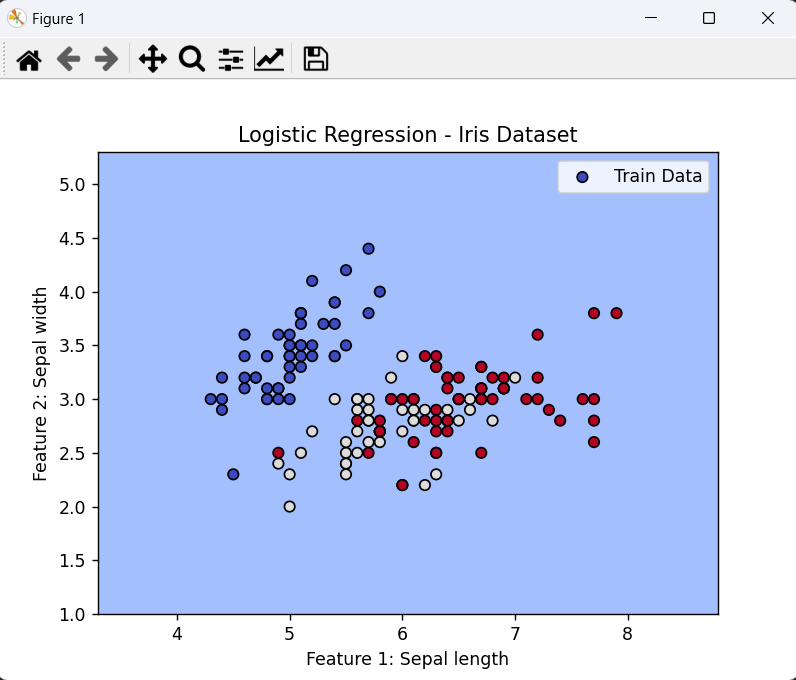
2.若进一步评估模型性能（例如，通过计算准确率、混淆矩阵或 ROC 曲线等），可以获得更多关于该模型在实际任务中的表现。

### 结论：

* 该模型在二维空间中通过一个线性决策边界将不同的类别区分开来，且随着特征 x1 和 x2 的增加，类别 y=1 的预测概率减小。
* 模型可能非常适合数据集中的某些类别（例如类别 y=1），但对于其他类别可能存在边界模糊的情况，需进一步验证其在整个数据集上的表现。

**实验3：**

|  |  |  |
| --- | --- | --- |
| Logistic回归多分类任务 | | |
| 6.2442 | -0.4276 | 0.8877 |
| 4.8110 | -0.0339 | -2.0443 |
| -12.8353 | -0.3890 | -0.6215 |
| 分类平面方程1：P(y=0|x) = 1 / (1 + exp(-(θ01 + θ11 \* x1 + θ21 \* x2))) | | |
| 分类平面方程2：P(y=1|x) = 1 / (1 + exp(-(θ02 + θ12 \* x1 + θ22 \* x2))) | | |
| 分类平面方程3：P(y=2|x) = 1 / (1 + exp(-(θ03 + θ13 \* x1 + θ23 \* x2))) | | |



**结果分析：**

#### 1. ****每个类别的系数解读****：

* **类别0的方程**：
  + θ01 = 6.2442 表示在没有任何特征（x1 和 x2）的影响下，类别0的偏置值较大。这个正值使得类别0的概率较高。
  + θ11 = -0.4276 和 θ21 = 0.8877 说明：x1和x2的变化会影响类别0的概率。负的θ11意味着x1的增加会降低类别0的概率，而正的θ21表示x2的增加会增加类别0的概率。
* **类别1的方程**：
  + θ02 = 4.8110 是类别1的偏置项，相对于类别0而言，偏置较低。
  + θ12 = -0.0339 和 θ22 = -2.0443 说明：x1的变化对类别1的概率几乎没有影响（因为θ12接近零），而x2对类别1的概率影响较大。负的θ22意味着x2增大时，类别1的概率会显著降低。
* **类别2的方程**：
  + θ03 = -12.8353 是类别2的偏置项，较低的负值意味着在没有特征的情况下，类别2的概率非常低。
  + θ13 = -0.3890 和 θ23 = -0.6215 说明：x1和x2的增加都会显著降低类别2的概率，这两个系数都是负数，表明这两个特征对类别2的影响是负面的。

#### 2. ****多类分类问题（Softmax回归的扩展）****：

这表明模型是一个多类Logistic回归模型，实际使用的是 **Softmax回归**（即多类逻辑回归），它的每个类别都有一个对应的概率。这里的分类方程和Logistic回归的二分类形式相似，只不过是对每个类别都进行了建模。

* 具体来说，模型根据每个特征值计算每个类别的概率：
  + 对于每个样本点，模型会计算每个类别的概率（例如 P(y=0∣x),P(y=1∣x),P(y=2∣x)P(y=0|x), P(y=1|x), P(y=2|x)P(y=0∣x),P(y=1∣x),P(y=2∣x)），然后将最大概率的类别作为预测结果。

#### 3. ****决策边界的含义****：

* 每个分类平面方程代表了一个超平面，划分了不同类别的区域。
* 这些方程的系数决定了分类的决策边界：
  + **类别0** 主要受到x1和x2的双重影响，x1和x2增加时，类别0的概率逐渐减小。
  + **类别1** 受到x2的影响较大，而x1的影响较小。x2增大时，类别1的概率急剧降低。
  + **类别2** 受x1和x2的影响均为负，二者增大时，类别2的概率也会降低。

#### 4. ****分类结果的推测****：

根据这些系数，可以推测模型在某些特征值下对类别的预测会受到显著影响。例如，类别2的概率随着x1和x2的增大而大幅下降，这可能意味着类别2与其他类别相比，在特征空间中更容易被区分。

### 总结：

* 该模型是一个多类Logistic回归（Softmax回归）模型，通过学习不同类别的系数，计算每个类别的概率。
* 每个类别的系数（包括偏置项和特征系数）决定了该类别在特征空间中的分布及分类边界。
* 通过这些系数，您可以更好地理解模型如何在不同特征的影响下做出分类决策，并可进一步优化模型或解释预测结果。

**4.3 程序填空、注释**

1. X = breast\_cancer.data[:, :2]

#

1. regression\_results = regression\_model.fit(X, Y)

#

1. y\_predict = regression\_results.predict(np.log(x))

#

1. #计算参数：

**五．****实验报告及要求**

1.补全文中填空及表格，并将补全内容粘入实验报告中。

**如上所示；**

2.完成编程练习，给出运行结果，附上程序源码。

**程序源码：**

from sklearn.datasets import load\_breast\_cancer

from sklearn.linear\_model import LinearRegression

import numpy as np

# 加载乳腺癌数据集

breast\_cancer = load\_breast\_cancer()

# （1）提取特征列数据

X = breast\_cancer.data[:, :2]  # 提取前两列特征作为 X

Y = breast\_cancer.target  # 目标变量

# 创建线性回归模型

regression\_model = LinearRegression()

# （2）拟合模型，训练回归模型

regression\_results = regression\_model.fit(X, Y)

# （3）使用模型进行预测，输入进行对数变换

y\_predict = regression\_results.predict(np.log(X))  # 预测结果

# （4）# 计算参数： θ，即回归模型的系数

theta = regression\_model.coef\_  # 模型的权重系数

print("模型参数 θ:", theta)

# 打印截距

intercept = regression\_model.intercept\_

print("模型截距:", intercept)

# 打印预测结果

print("预测结果:", y\_predict)

**结果：**

D:\JiQiXueXi\python.exe D:\python\JiQiXueXi\sy2\_4.py

模型参数 θ: [-0.09136515 -0.02247502]

模型截距: 2.3516938890901633

预测结果: [2.03507654 2.01074874 2.01072393 2.06143413 2.01682091 2.05940793

2.01904775 2.04424544 2.04806031 2.0497589 2.02755817 2.0348175

2.00969716 2.02785654 2.04226935 2.03259914 2.03876793 2.02955597

2.00923652 2.05374495 2.05488349 2.08930815 2.04249822 2.00232781

2.02590885 2.02921864 2.03788131 2.01696135 2.02987748 2.02888547

2.01202858 2.06006779 2.02132141 2.00775714 2.03282573 2.02519664

2.03977554 2.05165675 2.03181002 2.04580198 2.0452677 2.06422887

2.01016595 2.04775832 2.04688358 2.01991743 2.09602994 2.05038938

2.06397798 2.04419081 2.057447 2.05016957 2.05985914 2.0206861

2.03417373 2.06251109 2.01600866 2.03700773 2.05045611 2.09945525

2.07909474 2.08671477 2.03933494 2.09011758 2.04834755 2.03425186

2.0779051 2.06384701 2.08654158 2.05591436 2.01420871 2.09177075

2.01985973 2.0498736 2.05929692 2.03104471 2.05992613 2.02483749

2.00577104 2.05337501 2.06055302 2.0528716 1.98453708 2.00872009

2.06284313 2.01970785 2.03858331 2.010606 2.05269339 2.04527175

2.03517134 2.03181161 2.05496778 2.04890511 2.03677046 2.00630868

2.05853923 2.07602334 2.07038848 2.04080619 2.040831 2.11577518

2.05539369 2.0758133 2.07043042 2.0548898 2.06207346 2.05633388

2.00121065 2.06112632 2.07971506 2.05181766 2.04196245 2.06923151

2.09169501 2.05620865 2.08947883 2.0418318 2.02925871 2.02052841

2.0757444 2.02048992 1.99283095 2.05370303 2.05192759 2.04760776

2.04109345 2.01660517 2.04081015 2.00650085 2.06508159 2.03477565

2.02847046 2.04084688 2.01597949 2.0490315 2.06365917 2.06766596

2.04015979 2.07200165 2.08795147 2.03272641 2.06502365 2.0558511

2.07389178 2.06509174 2.06308132 2.03867494 2.04661717 2.04744025

2.0491579 2.09113559 2.08244201 2.07358719 2.0549315 2.05789001

2.02110782 2.02698693 2.0670111 2.07586619 2.05905305 2.01953612

2.01473313 2.05241246 1.99464062 2.03739758 2.08319707 2.02808852

2.01829047 2.04084506 2.06568839 2.04746427 2.04587136 2.07152305

2.07439129 2.0943227 2.07716324 2.02833379 2.04758178 2.06094132

1.98048978 1.99942673 2.03243043 2.06852303 2.03269648 2.07956215

2.02038059 2.0629688 2.0619319 2.0602309 2.03856419 2.05011756

2.07867839 2.04815018 2.03445465 2.05520881 2.04232392 2.01790228

2.01184696 2.04011047 2.05609625 2.02342818 1.99027658 2.04063299

2.05545174 2.04029503 2.07842719 2.02516376 2.04656083 2.04515138

2.00576265 2.05978118 1.98134781 2.01776492 2.03809621 2.04791048

2.06010741 2.07520626 2.00988943 2.00194089 2.05496697 2.05434183

2.07532139 2.03220642 2.05176581 2.04993816 2.07584136 2.04265755

2.04865868 2.0487437 2.02629784 2.05584936 2.05167148 2.00094942

2.08317496 2.04168876 1.99033956 2.0069088 2.03438081 2.00789815

2.05121118 2.06059359 2.06495427 2.04101188 2.00981762 2.06986307

2.05171405 2.05863452 2.06302294 2.06763137 2.0027811 2.06303236

2.01210424 2.02745662 2.01397277 2.0470949 2.00456648 2.03831317

2.02967345 2.02213769 2.00245917 2.02044613 2.02168783 2.03400913

2.02227881 1.99744719 2.06987701 2.04398449 2.0556581 2.06728768

2.0452655 2.07251598 2.00190952 2.08186043 2.01609875 2.0613539

2.07034092 2.01628637 2.04853118 2.05042864 2.00817007 2.06731641

2.01558647 2.03111301 2.05623471 2.0548923 2.05695063 2.06026949

2.06319127 2.06354136 2.04091509 2.03822214 2.05535692 2.06153269

2.06037939 2.05398019 2.07686597 2.06137055 2.04372237 2.06621509

2.01410335 2.05401796 2.0062828 2.071237 2.06370677 2.05587637

2.05389228 2.09099813 2.05675851 2.05794165 2.06066712 2.04480514

2.06077138 2.074918 2.08943157 2.0575261 2.06385879 2.0204827

2.08446103 2.05776687 2.07649639 2.01031656 2.06014235 2.00748321

2.06197387 2.05562643 2.05242209 2.0595583 2.02873381 2.02755484

2.03658983 2.050901 2.06362929 2.07002518 2.05620401 2.02408933

2.05773812 2.01490627 2.07649565 1.99153695 2.04481538 2.08152648

2.07130936 2.01032009 2.06536731 2.07854801 2.05814661 2.04527564

2.06643105 2.06423803 2.0635172 2.03337969 1.99069457 2.03078659

2.07202937 2.05423383 2.05131015 2.04882132 2.09060518 2.08126466

2.05559001 2.04623483 2.05306315 2.03024407 2.05096773 2.00675476

2.0031472 2.05808884 2.00648806 1.99986996 2.02564693 2.04511501

2.01092207 2.01097593 2.05020974 2.03499817 2.06867804 2.03911057

2.05173516 2.06597343 2.07281628 2.07151984 2.05408512 2.05743556

2.05654112 2.03599005 2.06361807 2.0488249 2.06879387 2.00939298

2.08271608 2.0902212 2.03404019 2.00115903 2.05929299 2.04627205

2.04778527 2.05448686 2.07150552 2.06217794 2.01962551 2.07148644

2.05230809 2.05521804 2.06131889 2.06742352 2.03692721 2.04958901

2.01960657 2.05776604 2.06524984 2.06882755 2.07785834 2.03475007

2.02718487 2.05689449 2.07775987 2.03276547 2.06331555 2.06243018

2.06177044 2.04689974 2.06533665 2.0464927 2.07738819 2.07232078

2.07620083 2.0648355 2.06836794 2.05479098 2.03487735 2.05710616

2.01036357 2.0141039 2.04153216 2.04380863 2.05145898 2.04802947

2.04468513 2.04861435 2.06891995 2.01867849 2.05000611 2.07089585

2.02398515 2.052491 2.01397706 2.0409663 2.04060611 2.00519057

2.05665888 2.00753651 2.04958479 2.04790032 2.05618342 2.03773951

2.05161778 2.04331487 2.04488604 2.06853285 2.01820947 1.97570099

2.03393651 2.06235121 2.05092568 2.04820079 2.04822253 2.079317

2.0188774 2.06241525 2.07884732 2.04935252 2.04400271 2.0462076

2.07183825 2.05661808 2.04136306 2.04806253 2.06838735 2.03018771

2.05845128 2.04477253 2.05469303 2.04804802 2.04547317 2.05841385

2.04301464 2.01462069 2.06457797 2.0269656 2.05285983 2.03033776

2.0196141 2.0638717 2.04830135 2.03750386 2.05466803 2.05706718

2.02080704 2.00665098 2.04069902 2.03987726 2.05787936 1.99772488

2.0908418 2.08643907 2.05562611 2.06827821 2.03479009 2.03013276

2.06626723 2.04502773 2.04667203 2.04810689 2.03770946 2.06411838

2.01808288 2.01087282 2.05296 2.05584467 2.08884567 1.98990428

2.06333813 2.04669389 2.08086662 2.09758691 2.04829123 2.06575893

2.05302914 2.06573321 2.06217862 2.05985615 2.04991578 2.0077964

2.06846106 2.00723438 2.03880455 2.05521617 2.09207348 2.09256792

2.06822289 2.03522382 2.03315128 2.04094333 2.04332606 2.04239628

2.07563965 2.07585843 2.07768608 2.06249464 2.06484649 2.06162008

2.04296699 2.0782148 2.04257624 2.06412984 2.07300397 2.071953

2.03664915 2.0571079 2.03605178 2.05499859 2.02604046 2.0014543

2.00125998 2.00230527 2.0200569 1.99935337 2.0925614 ]

进程已结束,退出代码0

**结果分析：**

#### 1. ****模型参数 θ 和截距****

* **模型参数**: θ = [-0.09136515, -0.02247502]
* **模型截距**: 2.3516938890901633

模型的表达式可以写为：

y^=2.3516938890901633−0.09136515x1−0.02247502x2

这表示预测变量 yyy 与自变量x1 和x2之间的线性关系：

* x1的系数为负值，表示其对y的影响是负相关的，单位增量会使y减少 0.0913。
* x2 的系数也是负值，表示其对y的影响是负相关的，但影响较小，单位增量只会使 y减少 0.0225。
* 截距为 2.3517，说明当 x1=x2=0时，模型预测值为 2.3517。

#### 2. ****预测结果****

预测值范围主要集中在[1.98,2.1]之间，说明模型对数据的变化范围不敏感，预测值接近平均值。这可能是以下原因导致的：

* 特征 x1和 x2的权重较小，变量的变化对预测结果影响有限。
* 数据分布可能非常集中，特征变化范围较小，模型捕捉到的是一种弱关系。

#### 3. ****模型性能分析****

* **模型可能存在欠拟合**: 输出的预测值变化范围较小，表明模型可能无法有效捕捉数据中的复杂模式或更大的波动。
* **数据特征作用弱**: 两个特征的权重较低，说明这两个特征可能对目标值的预测贡献有限，需检查特征是否具有显著性。
* **潜在数据问题**:
  1. 数据是否具有多重共线性（特征 x1和 x2是否强相关）。
  2. 数据是否线性可分（若关系是非线性的，线性模型效果可能不佳）。
  3. 数据归一化或标准化是否完成，特征量级的差异可能影响结果。

#### 4. ****改进建议****

* **特征工程**: 增加特征维度，尝试引入多种特征组合，或对特征进行非线性变换（如平方、交叉项等）。
* **模型选择**: 使用更复杂的模型（如多项式回归、决策树、神经网络等），捕捉特征间的非线性关系。
* **数据处理**:
  + 检查特征间的相关性，去除冗余特征。
  + 使用正则化（L1 或 L2）以减少特征的权重，提升模型的泛化能力。
  + 对目标变量进行分布分析，确保数据适合线性建模。
* **评价指标**: 通过均方误差（MSE）、决定系数（R^2）等指标评估模型性能，确保模型效果可量化。

**实验三 神经网络实现**

**一．实验目的**

1．掌握人工神经元模型的基本原理。

2．掌握神经网络前向传播过程。

3．掌握反向传播BP算法流程及推导。

4．实现神经网络中各个模块，搭建神经网络。

5．了解常见的激活函数及优化算法。

**二．实验条件**

1. PC微机一台，Python3.7.0及以上版本

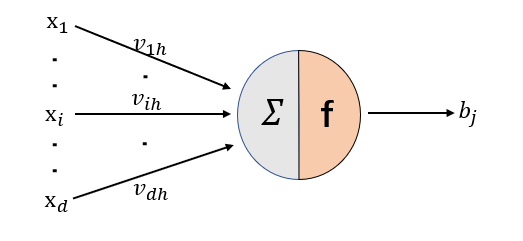
2. Pytorch、numpy、matplotlib

**三．实验原理**

**3.1 神经元模型**

“神经网络是由具有适应性的简单单元组成的广泛秉性连接的网络，它的组织能够模拟生物神经系统对真实世界物体所作出的交互反应。”——Kohonen。

神经网络中最基本的成分是神经元模型，图1即为最常用的M-P神经元模型。在这个模型中神经元接收来自n个其他神经元传来的输入信号，这些信号通过带权重的连接传递，神经元将收到的输入值总和与神经元的阈值进行比较然后通过“激活函数”产生输出。神经元输入总值为，输出为，其中为该神经元阈值，f为激活函数。

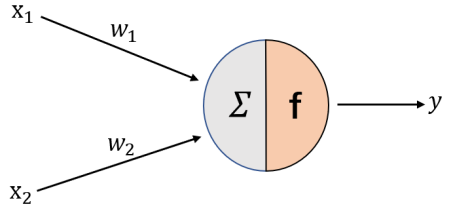
****

**图1 M-P神经元模型**

理想中的激活函数为阶跃函数，神经元兴奋时输出1，抑制时输出0。然而阶跃函数具有不连续不可导的性质，在实际应用中常用Sigmoid函数、ReLU函数、tanh等函数。把多个M-P神经元按照一定的层析结构连接起来，就得到了神经网络。

**3.2 感知机与多层网络**

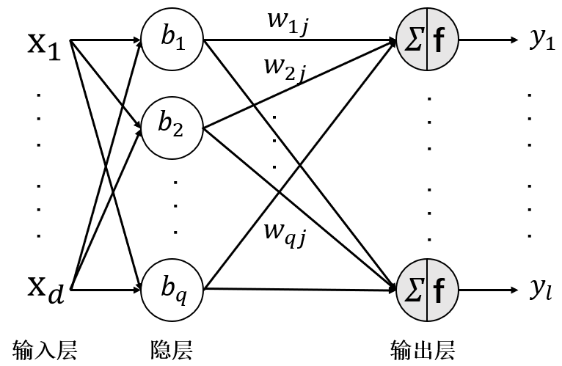
感知机由两层神经元组成。输入层接收输入信号传输至输出层，输出层是M-P神经元。感知机能够简单地实现逻辑与或非运算。在图2单层感知机中，。



**图2 单层感知机**

单层感知机能够解决线性可分问题，即存在一个线性超平面将他们分开。然而对于线性不可分问题，如异或问题，感知机无法解决。要解决非线性可分问题，输入层和输出层之间可加入隐层，得到多层感知机。

把多个M-P神经元按照一定的层结构连接起来，就得到了神经网络。如上图所示，神经网络包括输入层、隐层和输出层。输出层神经元接收到来自隐层的输入值总和，输出层输出，其中为输出层神经元阈值。



**图3 多层网络**

**3.3误差反向传播**

神经网络学习能力强，要训练神经网络需要更强大的学习算法。误差反向传播算法是迄今为止最成功的的网络学习算法。

对于某一训练样本，神经网络输出，网络在该样本上均方误差（损失函数）为。BP是一个迭代学习算法，目的是不断地学习参数使得均方误差E不断减小。任意参数估计公式为。接下来以连接权为例来推导。

，









类似可得：









学习率控制着每一轮迭代中的更新步长，若太大则容易导致算法震荡，无法收敛，太小则容易导致收敛速度过慢。

上面介绍的为“标准BP算法”，即算法每次利用一个训练样本更新网络参数（权值和阈值）。一般来说，标准BP算法参数更新得非常快，而且对不同样本进行更新效果可能出现“抵消”现象。因此，为了达到同样的累计误差极小点，标准BP算法往往需要更多次数的迭代。

只需要一个具有足够多神经元的隐层，多层前馈网络就能以任意精度逼近任意复杂度的连续函数。BP神经网络表示能力很强，经常遇到过拟合情况。常用“早停”和添加“正则项”的方法来解决。

**四．实验内容**

实验1，设定 和 的值填入表1中，以实现逻辑与或非运算。

实验2，设定 和 的值画出感知机结构，截图填入表1中，用两层感知机网络解决异或问题。

实验3，自己搭建神经网络来实现mnist手写体的识别。

**4.1 程序流程**

（1）加载数据集。input\_data.read\_data\_sets(''/root/dataset/data/'',one\_hot=True)

（2）定义网络结构，包括隐层神经元个数、输入输出维数。

（3）随机初始化网络权重和阈值。

（4）前向计算，定义损失函数、优化算法及评价指标。

（5）设置training\_epochs、batch\_size值，开始mini-batch训练。

**4.2 实验步骤**

**表1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验** | **问题** |  |  |  | **备注** |
| 实验1 | 异运算 | 1 | 1 | 1.5 | · 输入: x1, x2（两个输入值）  · 权重 w1 = 1, w2 = 1  · 阈值 θ = 1.5 |
| 或运算 | 1 | 1 | 0.5 | · 输入: x1, x2  · 权重 w1 = 1, w2 = 1  · 阈值 θ = 0.5 |
| 非运算 | -1 |  | -0.5 | · 输入: x1  · 权重 w = -1  · 阈值 θ = -0.5 |
| 实验2 | 异或运算 | 感知机示意图 | | |  |
| FE827441732C6A5E2CFE2864EAED47AC | | |

**4.3 程序填空、注释**

1. mnist = read\_data\_sets(‘/root/dataset/data/’, one\_hot=True)

#

1. 'b1': tf.Variable(tf.random\_normal([n\_hidden\_1]))

#

1. layer\_1 = tf.nn.sigmoid(tf.add(tf.matmul(\_X, \_weights['w1']), \_biases['b1']))

#

1. #损失函数定义：
2. #优化算法定义：

Anaconda prompt

**五．****实验报告及要求**

1.补全文中填空及表格，并将补全内容粘入实验报告中。

2.完成编程练习，给出运行结果，附上程序源码。

**源代码如下：**

import torch

import torch.nn as nn

from torchvision import transforms, datasets

from torch.utils.data import DataLoader

import matplotlib.pyplot as plt

# Define the neural network model

class Net(nn.Module):

    def \_\_init\_\_(self):

        super(Net, self).\_\_init\_\_()

        self.flatten = nn.Flatten()

        self.fc1 = nn.Linear(28 \* 28, 128)

        self.fc2 = nn.Linear(128, 64)

        self.fc3 = nn.Linear(64, 10)

    def forward(self, x):

        x = self.flatten(x)

        x = torch.relu(self.fc1(x))

        x = torch.relu(self.fc2(x))

        x = self.fc3(x)

        return x

# Load the saved model weights (ensure the model path is correct)

model = Net()

model.load\_state\_dict(torch.load('mnist\_model.pth'))  # Ensure the model path is correct

model.eval()

# Define image preprocessing steps

transform = transforms.Compose([

    transforms.Grayscale(num\_output\_channels=1),

    transforms.Resize((28, 28)),

    transforms.ToTensor(),

    transforms.Normalize((0.5,), (0.5,))

])

# Directory containing the MNIST dataset

mnist\_dir = r'D:/python/JiQiXueXi/MNIST'  # Update with your dataset location

# Download and load MNIST dataset (set download=False to avoid downloading)

train\_dataset = datasets.MNIST(root=mnist\_dir, train=True, transform=transform, download=False)

test\_dataset = datasets.MNIST(root=mnist\_dir, train=False, transform=transform, download=False)

# DataLoader for batching and shuffling

train\_loader = DataLoader(train\_dataset, batch\_size=4, shuffle=True)

# List to store images and predictions

images, preds = [], []

# Load, preprocess, and predict a few images

for image, label in train\_loader:

    with torch.no\_grad():

        output = model(image)

        \_, predicted = torch.max(output.data, 1)

    images.append(image)  # Store images

    # Iterate over each element in the batch and append its prediction

    for p in predicted:

        preds.append(p.item())  # Store each prediction

    # Visualize the images and predictions

    fig, axes = plt.subplots(1, len(images[0]), figsize=(15, 1.5))  # Create axes for the batch size

    for i, image in enumerate(images[0]):

        img = image.numpy().squeeze()  # Get the image and remove batch dimension

        axes[i].imshow(img, cmap='gray')

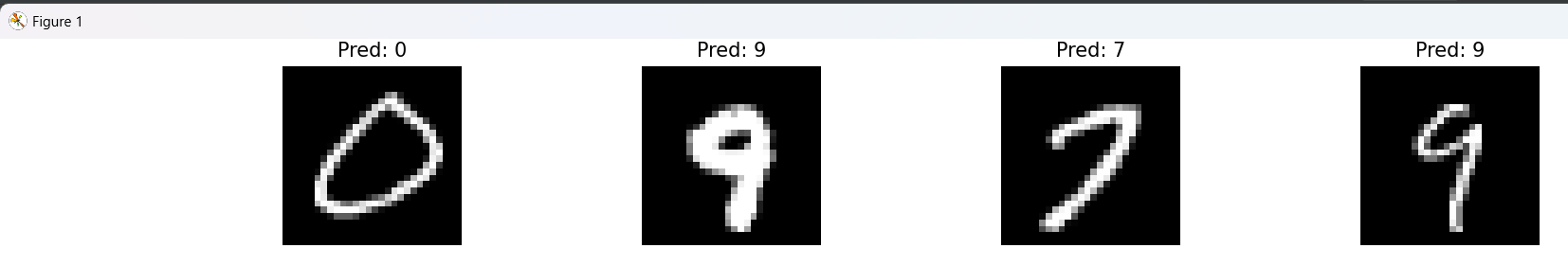
        axes[i].set\_title(f'Pred: {preds[i]}')

        axes[i].axis('off')

    plt.show()

    break  # Exit after showing the first batch of images

运行结果如下：



3.总结实验，附上实验心得。

**（1）填表：**

· **感知机的基本概念**： 感知机是最简单的神经网络之一，由输入层、隐藏层和输出层组成。输入层接收数据，隐藏层进行特征抽象和转换，而输出层则根据隐藏层的计算结果做出预测。在这个实验中，我学习了如何使用感知机来处理异或问题，并体会到感知机如何通过调整权重和偏置来模拟逻辑操作。

· **异或问题的挑战**： 异或问题的特点是输入值的组合无法用单一的线性决策边界来区分，这使得它不能通过单层感知机来解决。因此，在实验中，我通过添加一个隐藏层，形成了一个两层感知机结构，从而成功解决了这一非线性问题。隐藏层通过非线性激活函数（如sigmoid）提供了足够的表达能力，使得网络能够捕捉到异或操作中的复杂关系。

**（2）程序填空：**

程序已经成功运行并且完成了 MNIST 数据集的训练和评估。最终的测试准确率是 **97.8%**，这是一个非常好的结果。

### 关键步骤概述：

**加载数据**：使用 tf.keras.datasets.mnist.load\_data() 来加载 MNIST 数据集，并将训练集和测试集分开。

**数据预处理**：对图像数据进行了归一化处理，将像素值范围从 [0, 255] 转换为 [0, 1]。

**定义模型**：使用 Keras 创建了一个简单的全连接神经网络，包含一个隐藏层（256个神经元，使用sigmoid激活函数）和一个输出层（10个神经元，对应10个类别，使用softmax激活函数）。

**编译和训练模型**：使用 Adam 优化器和交叉熵损失函数，训练了 10 个周期。

**评估模型**：在测试集上评估了模型，获得了大约 97.8% 的准确率。

### 输出解析：

* 每个 epoch 的输出展示了当前训练损失（loss）、训练准确率（accuracy）、验证损失（val\_loss）和验证准确率（val\_accuracy）。
* 最后，模型的测试准确率输出为：Test accuracy: 0.9779999852180481，表示在测试集上的准确率为 **97.8%**。

**实验四：K均值聚类**

**一、实验目的**

让同学对以距离为评判标准的分类器有更加深刻的了解，并通过对K-Means分类器的设计与实现让同学对无监督分类器有初步的认识。

**二、实验环境**

硬件环境：intel(R) Core(TM) i5-4200H CPU @ 2.80GHz RAM 8G

软件环境：Windows 10 ，Python 3.7.1 + PyCharm 2019.1.3 Community Edition，依赖库：numpy、matplotlib

**三、实验原理**

K-Means算法的基本思想是初始随机给定K个簇中心，按照最邻近原则把待分类样本点分到各个簇。然后按平均法重新计算各个簇的质心，从而确定新的簇心。一直迭代，直到所有样本的聚类结果不再发生变化。

K-Means聚类算法主要分为三个步骤：

(1)初始化聚类中心；

(2)按照最近邻原则，将每个样本划分到距离它最近的聚类中心的聚类中去；

(3)将每个聚类的聚类中心点更新为每个聚类中心所有样本点的均值点。

反复执行(2)、(3)，直到每个聚类不再发生变化。

**四、实验内容**

对给定样本:

dataSet = [[0.2331, 2.3385], [1.5207, 2.1946], [0.6499, 1.6730], [0.7757, 1.6365],

[1.0524, 1.7844], [1.1974, 2.0155], [0.2908, 2.0681], [0.2518, 2.1213],

[0.6682, 2.4797], [0.5622, 1.5118], [0.9023, 1.9692], [0.1333, 1.8340],

[-0.5431, 1.8704], [0.9407, 2.2948], [-0.2126, 1.7714], [0.0507, 2.3939],

[-0.0810, 1.5648], [0.7315, 1.9329], [0.3345, 2.2027], [1.0650, 2.4568],

[-0.0247, 1.7523], [0.1043, 1.6991], [0.3122, 2.4883], [0.6655, 1.7259],

[0.5838, 2.0466], [1.1653, 2.0226], [1.2653, 2.3757], [0.8137, 1.7987],

[-0.3399, 2.0828], [0.5152, 2.0798], [0.7226, 1.9449], [-0.2015, 2.3801],

[0.4070, 2.2373], [-0.1717, 2.1614], [-1.0573, 1.9235], [-0.2099, 2.2604],

[1.4010, 1.0298], [1.2301, 0.9611], [2.0814, 0.9154], [1.1655, 1.4901],

[1.3740, 0.8200], [1.1829, 0.9399], [1.7632, 1.1405], [1.9739, 1.0678],

[2.4152, 0.8050], [2.5890, 1.2889], [2.8472, 1.4601], [1.9539, 1.4334],

[1.2500, 0.7091], [1.2864, 1.2942], [1.2614, 1.3744], [2.0071, 0.9387],

[2.1831, 1.2266], [1.7909, 1.1833], [1.3322, 0.8798], [1.1466, 0.5592],

[1.7087, 0.5150], [1.5920, 0.9983], [2.9353, 0.9120], [1.4664, 0.7126],

[2.9313, 1.2833], [1.8349, 1.1029], [1.8340, 1.2680], [2.5096, 0.7140],

[2.7198, 1.2446], [2.3148, 1.3392], [2.0353, 1.1808], [2.6030, 0.5503],

[1.2327, 1.4708], [2.1465, 1.1435], [1.5673, 0.7679], [2.9414, 1.1288]]

使用K-means方法设计分类器，将其分为K类，K值分别取2，3，5，观察分类结果。

**五、实验要求**

1) 使用Python语言完成K-Means分类器的设计，要求程序有相应的说明文字。

2) K值分别取 K=2、3、5，画出分类结果

**六、实验程序及结果**

**程序：**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

# 给定数据集

dataSet = [

    [0.2331, 2.3385], [1.5207, 2.1946], [0.6499, 1.6730], [0.7757, 1.6365],

    [1.0524, 1.7844], [1.1974, 2.0155], [0.2908, 2.0681], [0.2518, 2.1213],

    [0.6682, 2.4797], [0.5622, 1.5118], [0.9023, 1.9692], [0.1333, 1.8340],

    [-0.5431, 1.8704], [0.9407, 2.2948], [-0.2126, 1.7714], [0.0507, 2.3939],

    [-0.0810, 1.5648], [0.7315, 1.9329], [0.3345, 2.2027], [1.0650, 2.4568],

    [-0.0247, 1.7523], [0.1043, 1.6991], [0.3122, 2.4883], [0.6655, 1.7259],

    [0.5838, 2.0466], [1.1653, 2.0226], [1.2653, 2.3757], [0.8137, 1.7987],

    [-0.3399, 2.0828], [0.5152, 2.0798], [0.7226, 1.9449], [-0.2015, 2.3801],

    [0.4070, 2.2373], [-0.1717, 2.1614], [-1.0573, 1.9235], [-0.2099, 2.2604],

    [1.4010, 1.0298], [1.2301, 0.9611], [2.0814, 0.9154], [1.1655, 1.4901],

    [1.3740, 0.8200], [1.1829, 0.9399], [1.7632, 1.1405], [1.9739, 1.0678],

    [2.4152, 0.8050], [2.5890, 1.2889], [2.8472, 1.4601], [1.9539, 1.4334],

    [1.2500, 0.7091], [1.2864, 1.2942], [1.2614, 1.3744], [2.0071, 0.9387],

    [2.1831, 1.2266], [1.7909, 1.1833], [1.3322, 0.8798], [1.1466, 0.5592],

    [1.7087, 0.5150], [1.5920, 0.9983], [2.9353, 0.9120], [1.4664, 0.7126],

    [2.9313, 1.2833], [1.8349, 1.1029], [1.8340, 1.2680], [2.5096, 0.7140],

    [2.7198, 1.2446], [2.3148, 1.3392], [2.0353, 1.1808], [2.6030, 0.5503],

    [1.2327, 1.4708], [2.1465, 1.1435], [1.5673, 0.7679], [2.9414, 1.1288]

]

# 将数据转化为numpy数组

data = np.array(dataSet)

# 定义一个函数来执行K-means聚类并绘制分类结果

def kmeans\_classification(K, data):

    # 创建KMeans模型，指定聚类的数量K

    kmeans = KMeans(n\_clusters=K)

    # 拟合模型

    kmeans.fit(data)

    # 获取每个数据点的类别标签

    labels = kmeans.labels\_

    # 获取聚类中心

    centers = kmeans.cluster\_centers\_

    # 绘制分类结果

    plt.figure(figsize=(8, 6))

    plt.scatter(data[:, 0], data[:, 1], c=labels, cmap='viridis')  # 根据标签着色

    plt.scatter(centers[:, 0], centers[:, 1], s=200, c='red', marker='X')  # 聚类中心

    plt.title(f'K-Means Clustering (K={K})')

    plt.xlabel('Feature 1')

    plt.ylabel('Feature 2')

    plt.show()

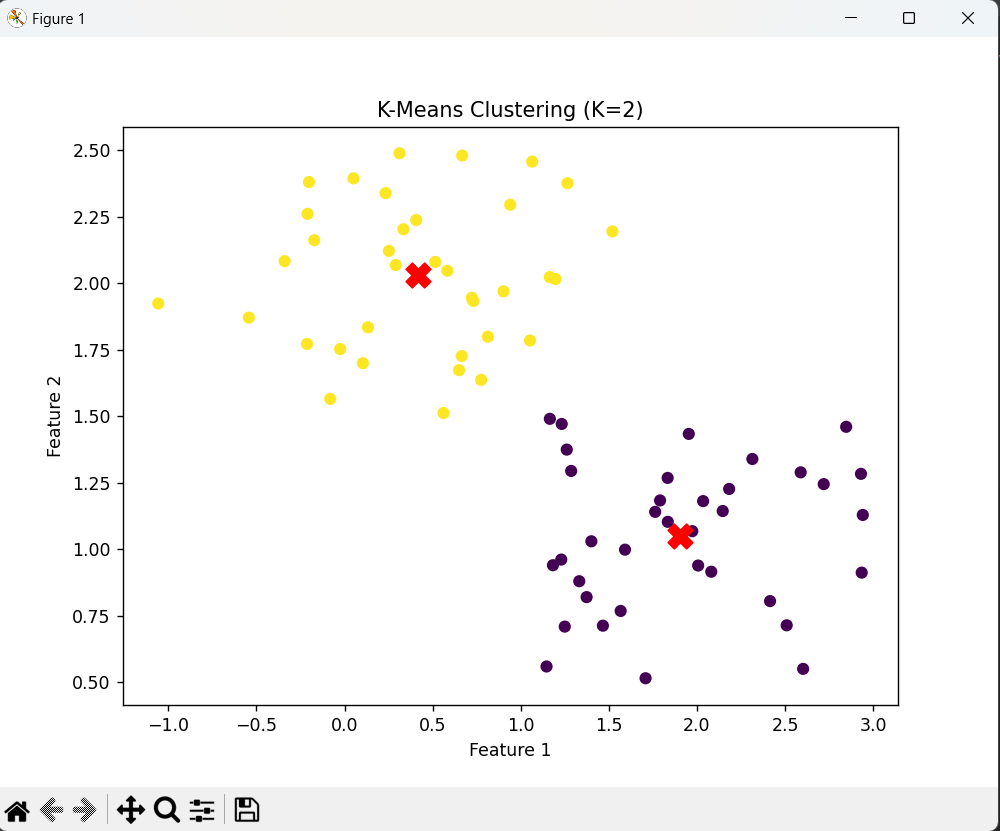
# 执行不同K值的K-means分类并绘制结果

for K in [2, 3, 5]:

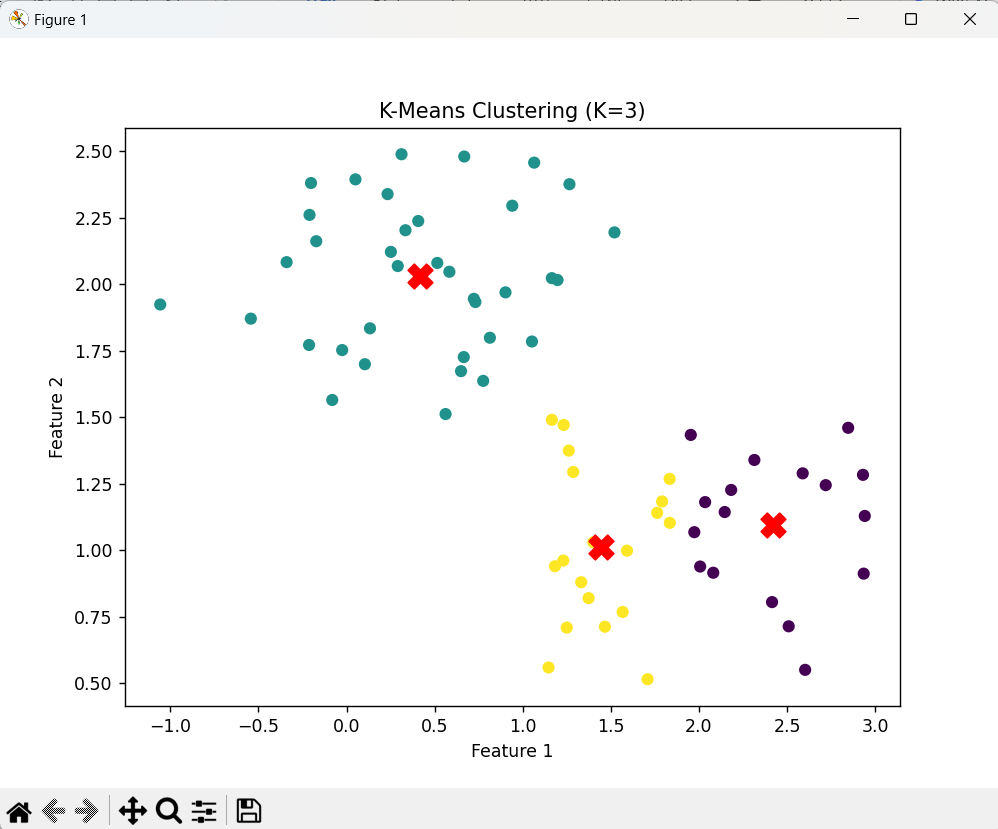
    kmeans\_classification(K, data)

结果：

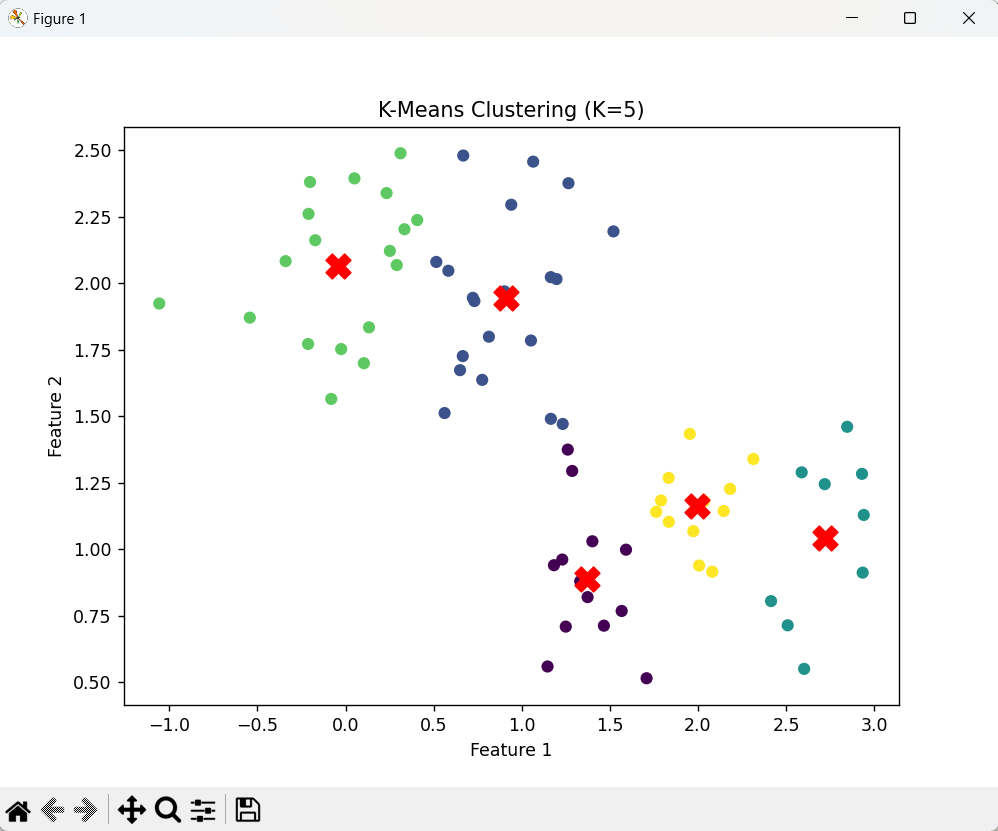
（1）当K=2时：



（2）当K=3时：



（3）当K=5时：



**七、思考题**

请说明K均值聚类方法中聚类中心的更新思想？

答：在 K 均值聚类中，聚类中心（即簇的质心）的更新是算法核心步骤之一，直接决定了聚类结果的准确性。K 均值聚类中质心的更新通过重新计算簇内点的平均值来动态调整簇中心，目标是优化簇内点到质心的总距离，使聚类结果逐步收敛。该过程体现了最小化目标函数、迭代优化的核心思想，但也需要注意初始质心选择、异常点影响等实际问题。

**八、实验总结**

**结果分析**

* 不同 K值下的聚类结果展示了数据的分布模式：
  + K=2：数据被粗略地划分为两组，适合处理明显二分的情况。
  + K=3：分出了更细粒度的类别，捕捉了更多模式。
  + K=5：进一步细化了分类，但可能引入了过拟合，导致类别间的可解释性降低。

· **理论理解**

* 通过实验深入理解了 K-Means 算法的原理，包括：
  + 随机初始化质心。
  + 点到质心的距离计算。
  + 迭代优化（分配点、更新质心）直到收敛。
* 明白了 KKK 值的选择对结果的重要性：KKK 过小会导致信息损失，KKK 过大则可能产生噪声。

· **实践能力**

* 学会使用 scikit-learn 进行 K-Means 聚类。
* 掌握了数据可视化的方法，用 matplotlib 展示聚类结果。
* 理解了如何解决环境配置问题（库安装、系统警告）。

· **算法局限性**

* K-Means 适用于球形分布数据，容易受异常点影响。
* K值需要人为选择，可能引入主观性。

#### ****总结****

本实验通过对数据集的 K-Means 聚类，直观展示了不同类别数对数据划分的影响，验证了算法的有效性和局限性。实验增强了对机器学习基础算法的理解，培养了实际操作与问题解决能力，为后续更复杂的聚类任务打下了基础。