《模式识别与机器学习 A》实验报告

实验题目: 多项式拟合正弦函数实验

学号: 2021110683

姓名: 徐柯炎

实验报告内容

1. 实验目的

掌握机器学习训练拟合原理（无惩罚项的损失函数）、掌握加惩罚项（L2范数）的损失函数优化、梯度下降法、理解过拟合、克服过拟合的方法(如加惩罚项、增加样本)

2. 实验内容

1. 生成数据，加入噪声；
2. 用高阶多项式函数拟合曲线（建议正弦函数曲线）；
3. 优化方法求解最优解（梯度下降）；
4. 用你得到的实验数据，解释过拟合。
5. 用不同数据量，不同超参数，不同的多项式阶数，比较实验效果。
6. 不许用现成的平台，例如 pytorch，tensorflow 的自动微分工具。建议实验编程环境：1）安装 anaconda 以及必要的工具包；2）建议采用 python 作为主编程语言，也可以根据个人需要和兴趣采用其它编程语言；3）可以基于实验室机器，也可以使用自己的便携式计算机上完成该实验。

3. 实验环境

Windows10; python3.9;PyCharm 2021.2.2

4. 实验过程、结果及分析（包括代码截图、运行结果截图及必要的理论支撑等）

（1） 多项式拟合的原理

在线性回归中，多项式拟合就是用类似泰勒级数，使用多项式来拟合正弦函数$sin(2\pi x)$ 。用公式来表示就是：

$$f_i(x_i,\theta)=\theta_0x_i^0+\theta_1x_i^1+\theta_2x_i^2+\ldots+\theta_mx_i^m,(0\le i\le num)$$

其中num为样本个数，m为阶数，θ_i为拟合的参数。

用矩阵表示：

$$f_\theta(X)=\theta X^T$$

其中

$$\theta=[\omega_0\quad \omega_1\ldots\omega_m]$$
$$X=[x_0^T\quad x_1^T\ldots x_{num}^T]$$

在这里通过足够的样本来训练可以得到合适的参数也就是θ，从而找到一般的学习规律。

（2） 梯度下降法原理

那么如何用合适的方法来得到多项式拟合的参数呢？最简单的方法就是梯度下降法。接下来将简要介绍一下梯度下降法的原理。

首先定义损失函数。损失函数就是一个自变量为算法的参数，函数值为误差值的函数。梯度下降就是找让误差值最小时候这个算法对应的参数。损失函数用

公式表示为：

$$J(\theta) = (f_\theta(X) - y)^2 + \frac{\lambda}{2}||\theta||_2$$

其中y为样本的真实标签（值），这里的损失函数为均方损失函数，λ为惩罚项，$||\theta||_2$为θ的 L2 范数。

接着初始化参数θ，一般采用随机化的方法初始化参数，并设置学习率α。

然后迭代更新参数：

$$\theta^t = \theta^{t-1} - \alpha\frac{\partial J(\theta^{t-1})}{\partial\theta^{t-1}}$$

其中t表示当前迭代的次数，$\frac{\partial J(\theta^{t-1})}{\partial\theta^{t-1}}$表示损失函数关于$\theta$的梯度。

接着检查终止条件：

a) 可以设置迭代次数的最大值。
b) 或者可以检查参数更新的幅度是否足够小，也就是设置一个阈值。

如果终止条件不满足，则进入下一个循环迭代参数。

这个过程将不断地更新参数，使损失函数或目标函数逐渐收敛到最小值或局部最小值。学习率α的选择和终止条件的设置都是梯度下降法的关键，它们会影响算法的性能和收敛速度。

（3） 实验过程

首先生成数据，加入噪声。

```
def generate_data(m=10, num=20, var=0.01, mean=0, left=0, right=1, plot=True):
    x = np.linspace(left, right, num)
    Y = np.sin(2 * np.pi * x)
    Y += np.random.normal(mean, var ** 0.5, num)
    X = np.zeros((m + 1, num))
    for i in range(m + 1):
        X[i] = x ** i
    if plot:
        plt.plot(x, Y, 'bo')
    Y = Y.reshape(1, num)
    # print(Y.shape)
    return X, Y
```

用高阶多项式函数拟合曲线（建议正弦函数曲线），正弦曲线如下。

```
def plot_sin():
    x = np.arange(0, 1, 0.01)
    y = np.sin(2 * np.pi * x)
    plt.plot(x, y, label='sin(2*pi*x)')
```

优化方法求解最优解（梯度下降），梯度下降函数如下。

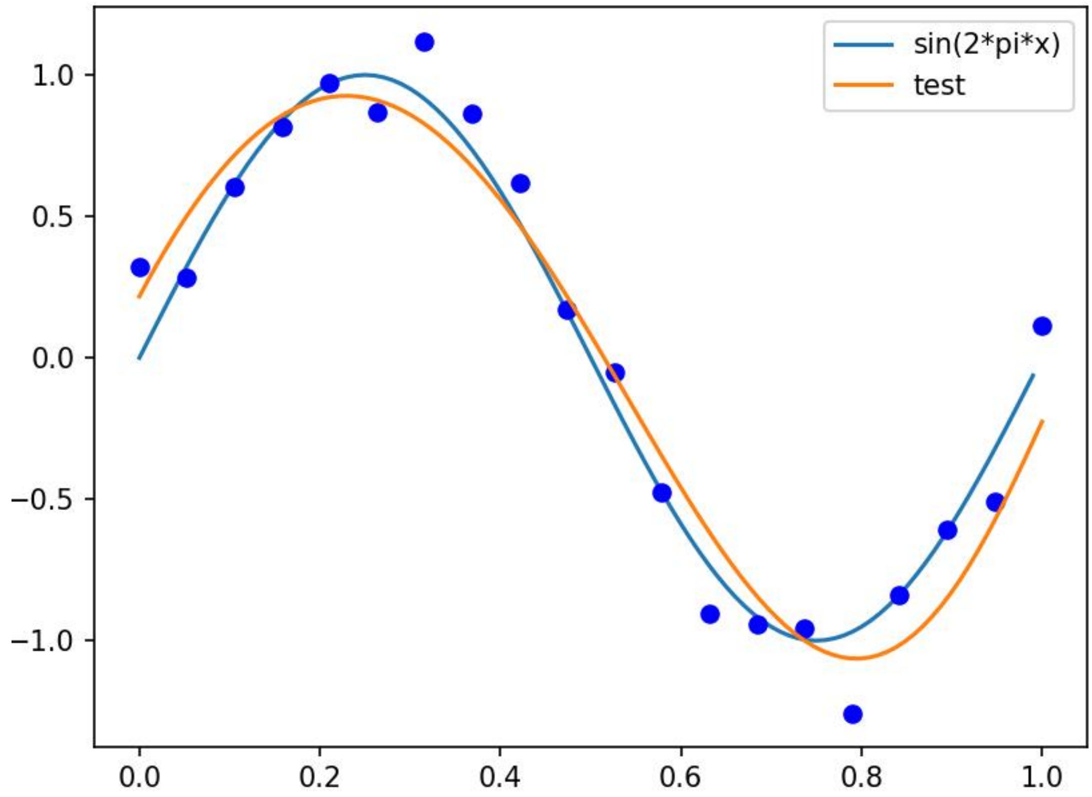
```python
def gradient_descent(x_train, y_train, alpha=0.01, m=10, iter=100000, lamuda=1, threshold=1e-5):
    """..."""
    # theta: shape(1, m+1)
    theta = np.random.rand(m + 1).reshape(1, m + 1)
    cost = 1e10
    for i in range(iter):
        gradient = gradient_theta(x_train, y_train, theta, lamuda)
        theta = theta - alpha * gradient
        if abs(cost - poly_cost(x_train, y_train, theta)) < threshold:
            cost = poly_cost(x_train, y_train, theta)
            print(f'iter:\t{i},\tcost:\t{cost}')
            break
        cost = poly_cost(x_train, y_train, theta)
        if i % 1000 == 0:
            print(f'iter:\t{i},\tcost:\t{cost}')
    print(f'拟合的参数theta分别为：', end='')
    for t in theta[0]:
        t = round(t, 5)
        print(t, end=' ')
    print()
    return theta
```

实验设置的参数如下：

```python
num = 20
m = 8
var = 0.01
iteration = 100000
learning_rate = 0.05
lamuda = 1
```

实验结果如下：

```
iter:   0,  cost:   25.327698444853173
iter:   1000,   cost:   0.7192789851502005
iter:   2000,   cost:   0.5964881163019027
iter:   3000,   cost:   0.5149430409322997
iter:   4000,   cost:   0.4528298598481535
iter:   5000,   cost:   0.4050973648576603
iter:   6000,   cost:   0.3685220565741726
iter:   7000,   cost:   0.340630115529994
iter:   8000,   cost:   0.3194844571499636
iter:   9000,   cost:   0.3035665720308429
iter:   10000,  cost:   0.2916872569239085
iter:   10063,  cost:   0.2910514729893614
拟合的参数theta分别为：0.21684 6.06423 -12.01066 -5.40744 3.84899 6.4258 5.32544 0.77581 -5.46511
```



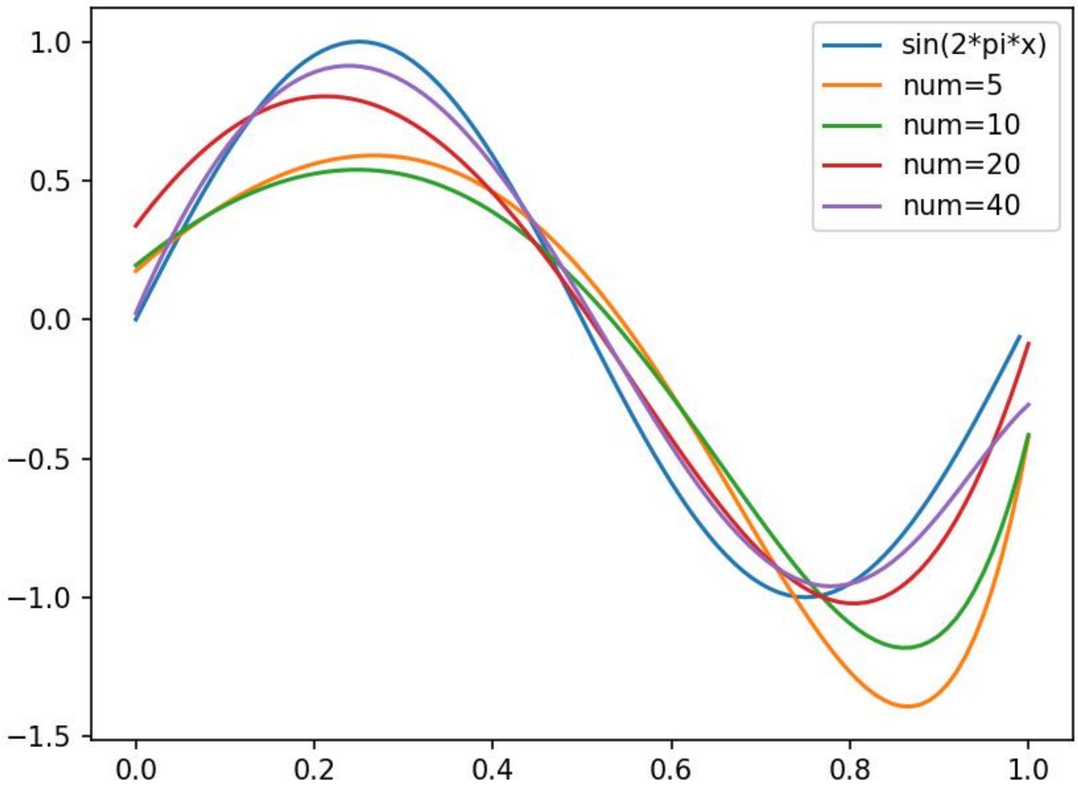
（4） 实验测试

a) 样本个数

测试代码如下：

```
def test_num():
    nums = [5, 10, 20, 40]
    plot_sin()
    for num in nums:
        x_train, y_train = generate_data(num=num, plot=False)
        theta = gradient_descent(x_train, y_train)
        test(theta, label='num='+f'{num}')
    plt.legend()
    plt.show()
```

测试结果如下：



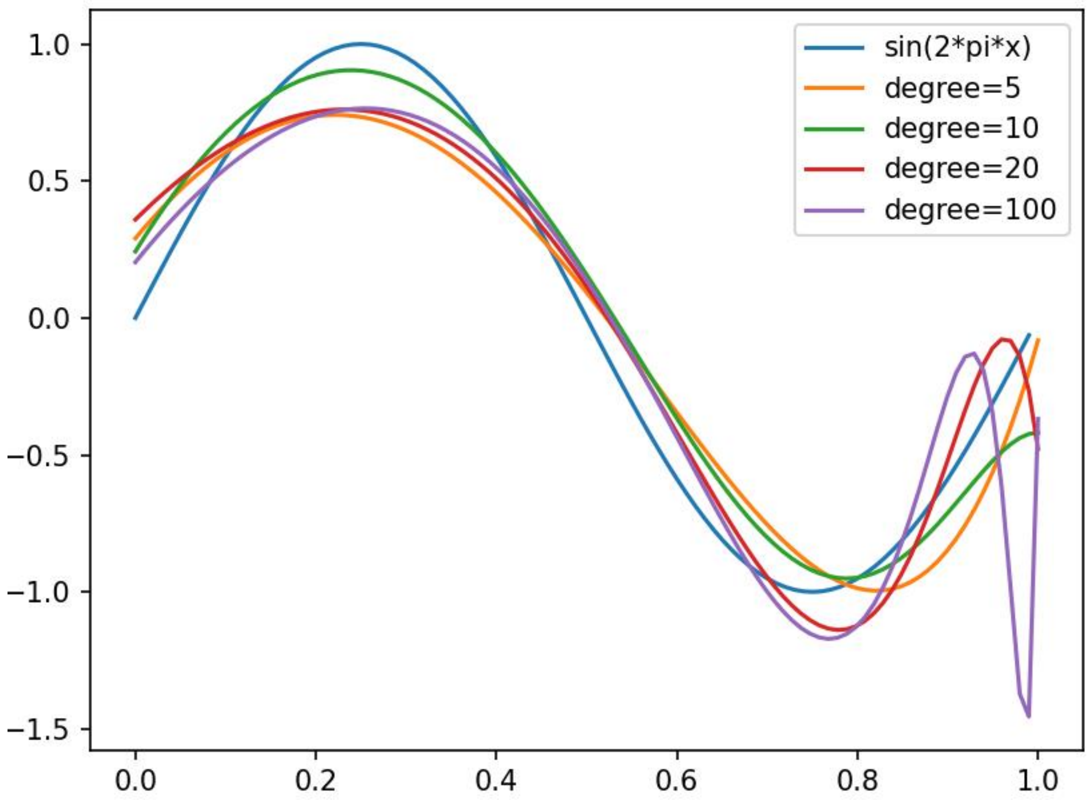
可以从上图看出，测试样本越少，越容易过拟合，也就是预测值和样本标签值几乎完全一致，但与实际情况不符。

b) 多项式阶数

测试代码如下：

```
def test_degree():
    plot_sin()
    degree = [5, 10, 20, 100]
    for m in degree:
        x_train, y_train = generate_data(m=m, plot=False)
        theta = gradient_descent(x_train, y_train, m=m)
        test(theta, label='degree=' + f'{m}')
    plt.legend()
    plt.show()
```

测试结果如下：



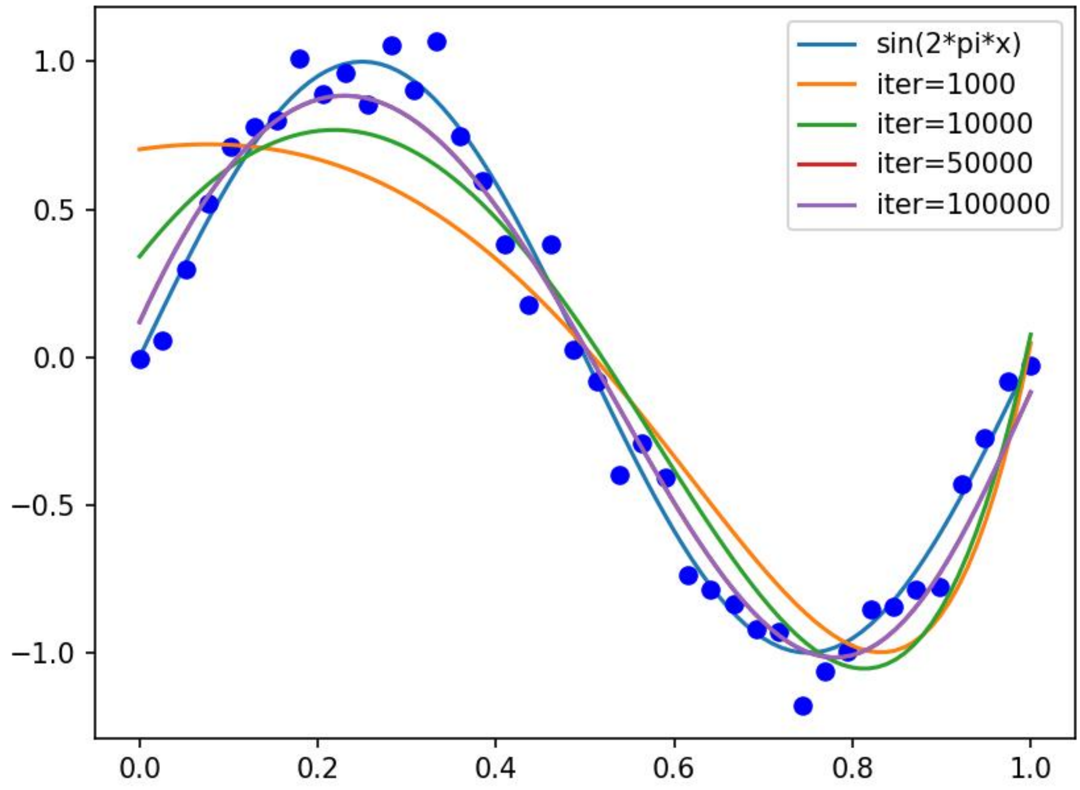
可以从上图看出，多项式阶数越大，曲线尾部越容易出现震荡的现象，也就是越容易过拟合，与正弦函数差距较大。

c) 迭代次数

测试代码如下：

```
def test_iter():
    num = 20
    m = 8
    var = 0.01
    iteration = [1000, 10000, 50000, 100000]
    learning_rate = 0.01
    lamuda = 1
    plot_sin()
    x_train, y_train = generate_data(m=m, var=var, num=num)
    for iter in iteration:
        theta = gradient_descent(x_train, y_train, m=m, lamuda=lamuda, iter=iter, alpha=learning_rate)
        test(theta, label='iter='+f'{iter}')
    plt.legend()
    plt.show()
```

测试结果如下：



可以从上图看出，迭代次数越多，拟合的效果越好，次数越少，越容易出现欠拟合的现象。

d) 学习率

测试代码如下：

```
def test_alpha():
    num = 20
    m = 8
    var = 0.01
    iteration = 100000
    learning_rate = [0.01, 0.02, 0.03, 0.04, 0.05]
    lamuda = 1
    plot_sin()
    x_train, y_train = generate_data(m=m, var=var, num=num)
    for alpha in learning_rate:
        theta = gradient_descent(x_train, y_train, m=m, lamuda=lamuda, iter=iteration, alpha=alpha)
        test(theta, label='alpha='+f'{alpha}')
    plt.legend()
    plt.show()
```

测试结果如下：

```
iter:   27659,  cost:   0.307255857539472
拟合的参数theta分别为：0.18091 4.95915 -9.76466 -4.02587 2.30317 4.95576 3.45373 1.37758 -3.36702
iter:   21183,  cost:   0.20551929519315393
拟合的参数theta分别为：0.09573 6.12112 -12.29798 -3.92209 3.20061 5.9809 4.99041 1.09445 -5.22262
iter:   16666,  cost:   0.1709551912963152
拟合的参数theta分别为：0.05949 6.58422 -13.10705 -4.4319 3.52996 7.31085 5.40655 0.40407 -5.73014
iter:   14087,  cost:   0.15335163187759743
拟合的参数theta分别为：0.03872 6.8438 -13.55615 -4.65481 3.6817 7.58948 6.10045 0.38757 -6.41379
```

可以从上图看出， 学习率越高，迭代次数越少，越容易收敛；但是在测试中，如果学习率过大的话，导致梯度较大，导致梯度爆炸，这会导致我们的 loss 在训练时变为 nan，也称之为数据溢出。

5. 实验总体结论

（1） 梯度下降法的优点与缺点：

a) 优点：

广泛应用： 梯度下降法是一种通用的优化方法，可用于解决各种机器学习和数据分析问题，包括线性回归、逻辑回归、神经网络等。

速度快： 梯度下降法通常在迭代的早期就可以快速收敛到局部最优解附近，特别是在大规模数据集上。

易于实现： 梯度下降法的算法相对简单，容易实现，并且适用于各种机器学习框架和编程语言。

b) 缺点：

选择学习率困难： 学习率是梯度下降法的关键超参数，选择不当的学习率可能导致算法不收敛或收敛速度过慢。需要进行调试和调优。

可能陷入局部最优解： 对于非凸损失函数，梯度下降法可能会陷入局部最优解，而无法找到全局最优解。这取决于初始参数和学习率的选择。

可能受局部梯度幅度影响： 如果损失函数在某些方向上的梯度较小，而在其他方向上的梯度较大，梯度下降法可能会受到梯度幅度不平衡的影响，导致收敛困难。完整实验代码

（2） 调试参数的一些结论

a) 样本个数

测试样本越少，越容易过拟合，也就是预测值和样本标签值几乎完全一致，但与实际情况不符。

b) 多项式阶数

多项式阶数越大，曲线尾部越容易出现震荡的现象，也就是越容易过拟合，与正弦函数差距较大。

c) 迭代次数

迭代次数越多，拟合的效果越好，次数越少，越容易出现欠拟合的现象。

d) 学习率选择

学习率越高，迭代次数越少，越容易收敛；但是在测试中，如果学习率过大的话，导致梯度较大，导致梯度爆炸，这会导致我们的 loss 在训练时变为 nan，也称之为数据溢出。

6. 完整实验代码

```python
# --*-- coding:utf-8 --*--
"""
作者：徐柯炎
日期：2023年10月05日
"""
import numpy as np
```

```python
import matplotlib.pyplot as plt


# 画出 sin 函数
def plot_sin():
    x = np.arange(0, 1, 0.01)
    y = np.sin(2 * np.pi * x)
    plt.plot(x, y, label='sin(2*pi*x)')


# 生成数据（高斯噪声）
def generate_data(m=8, num=20, var=0.01, mean=0, left=0, right=1, plot=True):
    x = np.linspace(left, right, num)
    Y = np.sin(2 * np.pi * x)
    Y += np.random.normal(mean, var ** 0.5, num)
    X = np.zeros((m + 1, num))
    for i in range(m + 1):
        X[i] = x ** i
    if plot:
        plt.plot(x, Y, 'bo')
    Y = Y.reshape(1, num)
    # print(Y.shape)
    return X, Y


# 求解梯度
def gradient_theta(x_train, y_train, theta, lamuda):
    gradient = np.dot(np.dot(theta, x_train) - y_train, x_train.T) + np.sum(lamuda * theta)
    return gradient


# 损失函数
def poly_cost(x_train, y_train, theta):
    return 0.5 * np.sum((np.dot(theta, x_train) - y_train) ** 2)


def least_square(x_train, y_train):
    theta = np.linalg.inv(np.dot(x_train, x_train.T)).dot(x_train).dot(y_train.T).reshape(1, -1)
    # print(np.linalg.inv(np.dot(x_train, x_train.T)))
    # print(theta)
```

```
    return theta


# 梯度下降
def gradient_descent(x_train, y_train, alpha=0.01, m=8, iter=100000, lamuda=1, threshold=1e-5):
    """
    :param threshold
    :param m: degree
    :param x_train: shape(m+1, num)
    :param y_train: shape(1, num)
    :param alpha: learning rate
    :param iter: iteration
    :param lamuda: penalty term
    :return: theta
    """
    # 参数初始化
    # theta: shape(1, m+1)
    theta = np.random.rand(m + 1).reshape(1, m + 1)
    cost = 1e10
    for i in range(iter):
        # 求解当前梯度
        gradient = gradient_theta(x_train, y_train, theta, lamuda)
        # 更新参数
        theta = theta - alpha * gradient
        # 结束迭代的条件
        if abs(cost - poly_cost(x_train, y_train, theta)) < threshold:
            cost = poly_cost(x_train, y_train, theta)
            print(f'iter:\t{i},\tcost:\t{cost}')
            break
        # 计算损失
        cost = poly_cost(x_train, y_train, theta)
        # if i % 1000 == 0:
            # print(f'iter:\t{i},\tcost:\t{cost}')
    print(f'拟合的参数 theta 分别为: ', end='')
    for t in theta[0]:
        t = round(t, 5)
        print(t, end=' ')
    print()
    return theta


```

```python
# 测试拟合效果
def test(theta, num=100, left=0, right=1, label="test"):
    de = theta.shape[1]
    # print((de, num))
    x = np.linspace(left, right, num).reshape(1, -1)
    X = np.zeros((de, num))
    for i in range(de):
        X[i] = x ** i
    y = np.dot(theta, X)
    # print(x.shape, y.shape)
    plt.plot(x[0], y[0], label=label)


# 测试学习率变化
def test_alpha():
    num = 20
    m = 8
    var = 0.01
    iteration = 100000
    learning_rate = [0.01, 0.02, 0.03, 0.04]
    lamuda = 1
    plot_sin()
    x_train, y_train = generate_data(m=m, var=var, num=num)
    for alpha in learning_rate:
        theta = gradient_descent(x_train, y_train, m=m, lamuda=lamuda, iter=iteration, alpha=alpha)
        test(theta, label='alpha='+f'{alpha}')
    plt.legend()
    plt.show()


# 测试不同迭代次数
def test_iter():
    num = 40
    m = 8
    var = 0.01
    iteration = [1000, 10000, 50000, 100000]
    learning_rate = 0.01
    lamuda = 1
    plot_sin()
    x_train, y_train = generate_data(m=m, var=var, num=num)
    for iter in iteration:
```

```python
        theta = gradient_descent(x_train, y_train, m=m, lamuda=lamuda, iter=iter, alpha=learning_rate)
        test(theta, label='iter='+f'{iter}')
    plt.legend()
    plt.show()


# 测试训练样本个数
def test_num():
    nums = [5, 10, 20, 40]
    plot_sin()
    for num in nums:
        x_train, y_train = generate_data(num=num, plot=False)
        theta = gradient_descent(x_train, y_train)
        test(theta, label='num='+f'{num}')
    plt.legend()
    plt.show()


# 测试多项式阶数
def test_degree():
    plot_sin()
    degree = [5, 10, 20, 100]
    for m in degree:
        x_train, y_train = generate_data(m=m, plot=False)
        theta = gradient_descent(x_train, y_train, m=m)
        test(theta, label='degree=' + f'{m}')
    plt.legend()
    plt.show()


# 测试惩罚项
def test_lamuda():
    plot_sin()
    m = 20
    lamudas = [0.1, 0.5, 1, 2]
    iter = 100000
    threshold = 1e-8
    num = 20
    x_train, y_train = generate_data(num=num, m=20)
    for lamuda in lamudas:
        theta = gradient_descent(x_train, y_train, lamuda=lamuda, threshold=threshold, m=20, iter=iter)
```

```python
        test(theta, label='lamuda=' + f'{lamuda}')
    plt.legend()
    plt.show()


# 测试主函数
def test_main():
    num = 20
    m = 8
    var = 0.01
    iteration = 100000
    learning_rate = 0.05
    lamuda = 1
    plot_sin()
    x_train, y_train = generate_data(m=m, var=var, num=num)
    # theta = least_square(x_train, y_train)
    theta = gradient_descent(x_train, y_train, m=m, lamuda=lamuda, iter=iteration, alpha=learning_rate)
    test(theta)
    plt.legend()
    plt.show()


if __name__ == '__main__':
    test_alpha()
```

7. 参考文献

[详解梯度下降法（干货篇）]