哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2020 年春季学期

计算学部《机器学习》课程

Lab 1 实验报告

姓名	刘璟烁
学号	
班号	
电子邮件	
手机号码	

目录

一、实验目的

掌握最小二乘法求解（无惩罚项的损失函数）、掌握加惩罚项（2 范数）的损失函数优化、梯度下降法、共轭梯度法、理解过拟合、克服过拟合的方法(如加惩罚项、增加样本)。

二、实验要求及实验环境

实验要求

1. 生成数据，加入噪声；
2. 用高阶多项式函数拟合曲线；
3. 用解析解求解两种 loss 的最优解（无正则项和有正则项）
4. 优化方法求解最优解（梯度下降，共轭梯度）；
5. 用你得到的实验数据，解释过拟合。
6. 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。
7. 语言不限，可以用 matlab，python。求解解析解时可以利用现成的矩阵求逆。梯度下降，共轭梯度要求自己求梯度，迭代优化自己写。不许用现成的平台，例如 pytorch，tensorflow 的自动微分工具。

实验环境

操作系统：MacOS 10.15.7
python 版本：Python 3.8.3

三、设计思想及算法实现

算法设计原理

1.样本生成算法：

在区间[0,1]间取若干个（10、50、100 等）均匀分布的样本点，并利用三角函数sin $2\pi x$生成对应的标记，并为标记增加均值为 0，方差为 0.2（在观察过拟合时调为 0.5，以使过拟合现象更为明显）的高斯噪声，最后输入和输出数据分别记为列向量 X、列向量 T。

2.解析解（无惩罚项）

主要思想为利用多项式对训练集中的样本点进行拟合，产生模式 $y(x, w)$用以预测样本对应的标记值：$y(x, w) = w_0 + w_1 x + \cdots + w_m x^m = \sum_{i=0}^{m} w_i x^i$，多项式的阶数 m 为超参数，需要自主设定；而参数 **w** 经过对损失函数求极值期望得到对训练集拟合效果最好的模式 y。

根据题目要求，我们利用最小二乘法进行损失的度量，损失函数可表示为：

$$E(w) = \frac{1}{2}\sum_{i=1}^{n}(y(x, w) - t)^2$$

我们可以利用生成的 X，T，生成关于样本属性的矩阵 $\mathbf{X_h}$，该矩阵中每个样本以 x^0、x、x^1...x^m 的行向量形式存在。则损失函数可以向量化为如下的形式：

$$E(w) = \frac{1}{2}(X_h W - T)^T (X_h W - T)$$

将括号拆开可以得到：

$$E(w) = \frac{1}{2}(w^T X_h^T X_h w - 2w^T X_h^T T + T^T T)$$

损失函数对参数向量 **w** 求偏导得：

$$\frac{\partial E}{\partial w} = X_h^T X_h w - X_h^T T$$

令该式为 0，得到参数 **w** 的解析解：

$$w = (X_h^T X_h)^{-1} X_h^T T$$

3.解析解（加入惩罚项）

理论上，可以通过为参数 **w** 的范数乘以一个权值加入到损失函数中，从而减少过拟合带来的 **w** 中各维度值偏大、模型过于复杂的问题。此时损失函数可以表示为：

$$E(w) = \frac{1}{2}\sum_{i=1}^{n}(y(x,w)-t)^2 + \frac{\lambda}{2}||w||^2$$

则该式对于参数 **w** 的偏导对应的向量为：

$$\frac{\partial E}{\partial w} = X_h^T X_h w - X_h^T T + \lambda w$$

令该式为 0，可以得到 **w** 为：

$$w = (X_h^T X_h + \lambda)^{-1} X_h^T T$$

4.梯度下降法（无惩罚项）

根据定义，梯度下降法是一种求取能使得损失函数近似最小的参数的最优化算法，使用梯度下降法可以找到一个函数的局部极小值。我们知道某一点梯度的反方向是函数值下降最快的方向，通过设定一个步长，沿向函数上当前点对应梯度（或者是近似梯度）的反方向的规定步长距离点进行迭代搜索，也就是每次迭代为向量参数 w 减去一个步长和梯度的乘积，我们知道损失函数关于变量 w 的偏导数如下（可直接利用前面解析解中的部分结果）：

$$\frac{\partial E}{\partial w} = X_h^T X_h w - X_h^T T$$

则每一次迭代，我们为 w 给予一个变化量，不妨设步长（学习率）为α，则迭代公示：

$$w = w - \alpha \frac{\partial E}{\partial w}$$

5.梯度下降法（加入惩罚项）

同样的，在损失函数中加入关于 w 的惩罚项，损失函数的值将在某种程度上受到 w 的范数的影响，从而减少模式对训练数据过拟合的程度。前面我们已经得出损失函数的表达式：

$$E(w) = \frac{1}{2}\sum_{i=1}^{n}(y(x,w) - t)^2 + \frac{\lambda}{2}||w||^2$$

以及此时损失函数关于 w 的偏导：

$$\frac{\partial E}{\partial w} = X_h^T X_h w - X_h^T T + \lambda w$$

根据梯度下降法的定义，我么只需要将变换后的偏导式代入迭代方程即可。

6.共轭梯度法（包含及不包含惩罚项）

根据维基百科的定义，共轭梯度法主要是用于求解形如 Ax = b 的线性方程组，在本实验中我们通过让损失函数 E 关于 w 的偏导等于 0 得到类似的线性方程组，根据 E 中含有惩罚项与否，线性方程组形式会发生细微变化。共轭梯度法区别于梯度下降法的一点是，后者每一次迭代的增量是损失函数 E 关于整个向量 w 的偏导，这不能保证 w 中的每一个维度都靠近最优解；而共轭梯度法每一次迭代改变 w 中一个维度的分量，使其逼近最优解。具体思想是每次计算向量 b 与

Aw 的残差 r，并以此来构造每一步迭代时的搜索方向 p，迭代的步长是通过残差 r、搜索方向 p 和系数矩阵 A 来确定的，每次迭代更新残差 r、p，以及参数向量 w，具体的，步长的计算函数如下：

$$\alpha = \frac{r_k^T r_k}{p_k^T A p_k}$$

而更新搜索方向需要用到系数β，其计算公式为：

$$\beta = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$$

算法实现

1.含惩罚项的解析解

我们直接通过 5 前面求出的 E 对参数向量 w 的偏导为零时 w 的式子，

$$w = (X_h^T X_h + \lambda I)^{-1} X_h^T T$$

调用 numpy 中求逆、点积等方法得出参数向量 w。

```
def analical_solution_withPun(X,T,lambd):
    K = np.linalg.inv(np.dot(X.T,X)\
                      +lambd*np.identity(X.ndim))
    W = np.dot(np.dot(K,X.T),T)
    return W
```

2.含惩罚项的梯度下降法

套用算法原理中 dw 的函数式：

$$\frac{\partial E}{\partial w} = X_h^T X_h w - X_h^T T + \lambda w$$

进行合理选择的若干次迭代，可以得到一个金丝最优的参数 w，主要迭代过程如下：

```
for i in range(num_ite):
    K = np.dot(X_head.T,X_head)
    dW = np.dot(K+lambd*np.identity(len(W)),W) \
        -np.dot(X_head,T)
    W = W - alpha * dW
    return W
```

源代码中为计算迭代过程中训练误差以及测试集中误差的变化，增加其他内容，详见 gradientDecent.py。不含惩罚项的迭代过程基本一致，只需将 dw 根据偏导的函数式适当调整即可。

3.含惩罚项的共轭梯度法

实现方法按照上述原理，先将偏导等于 0 的方程套入 AX=b 每个量中，并对残差和搜索方向进行初始化，随后开始迭代，每次迭代通过残差 r、搜索方向 p 和系数矩阵 A 计算迭代步长，对参数 w（也就是线性方程组中的 x）、残差 r 进行更新，并以此来构造下一步迭代时的搜索方向 p，当残差的值小于某一定值（超参数）的时候结束迭代。

```
def co_gradient_withPun(X,T,W):
    A = np.dot(X.T, \ X)+lambd*np.identity(X.T.shape[0])
```

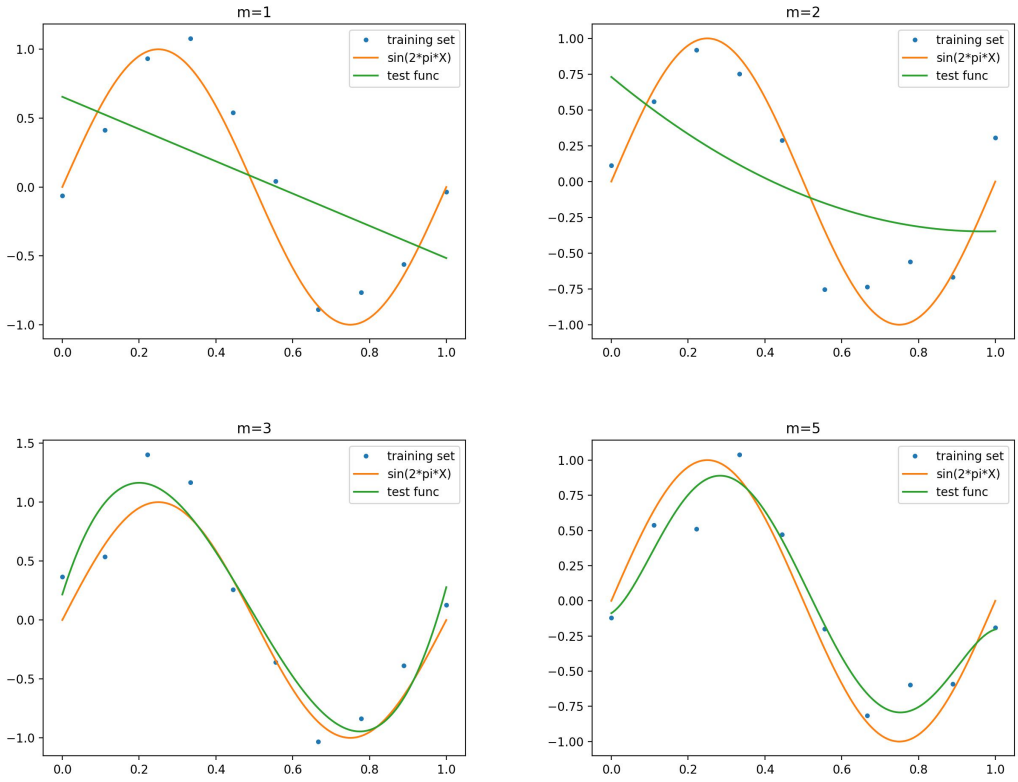
```
    b = np.dot(X.T, T)
    X = W #上面三行将参数代入线性方程组
    r = b - np.dot(A, X)
    p = r
    while True:
        alpha = np.dot(r.T, r)/np.dot(np.dot(p.T, A), p) #计算步长
        X = X + alpha * p #对参数 W，也就是这里的 X 进行迭代
        r_next = r - alpha * np.dot(A, p) #计算下一个残差，后面还用到当前
的残差，故需要保留
        if np.linalg.norm(r_next) < 0.000001:
            break #当残差小于某个值我们认为拟合效果足够，停止迭代
#计算搜索方向式子中需要的参数
        p = r_next + beta * p #更新搜索方向
        r = r_next #更新残差
    return X
```

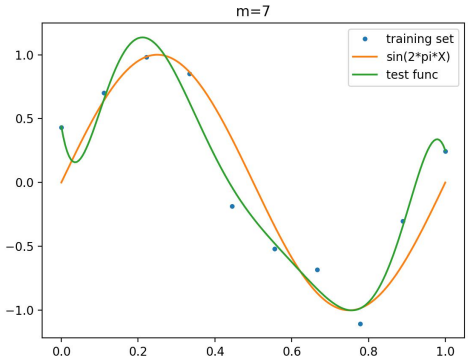
四、实验结果分析

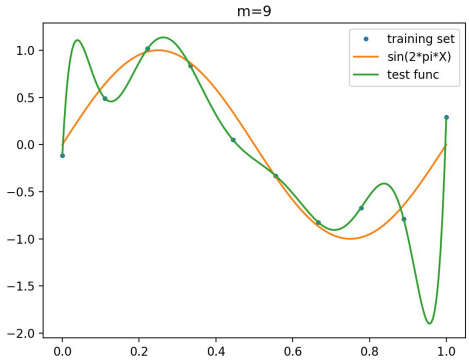
1.解析解（无惩罚项）

(1).固定样本数，设置不同的阶数

分别设置 m 的阶数为 1、2、3、5、7、9，并为了更加观察到老师 ppt 中出现的过拟合现象，把训练样本数固定为 10 个，并使用 100 个测试样本点描绘生成参数 w 对应的曲线。观察到拟合曲线如下：



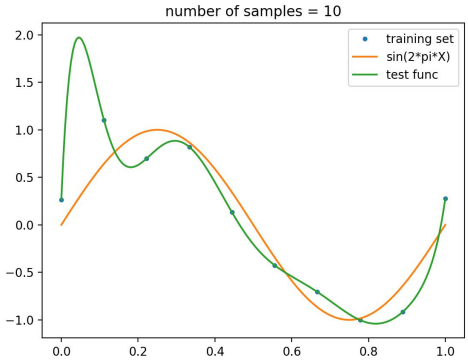


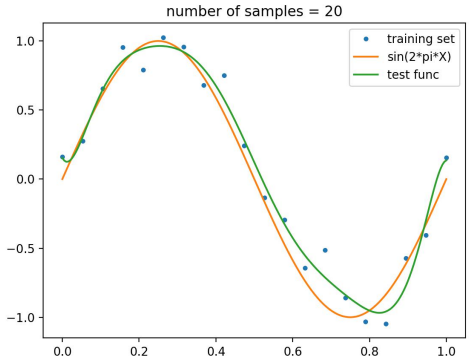


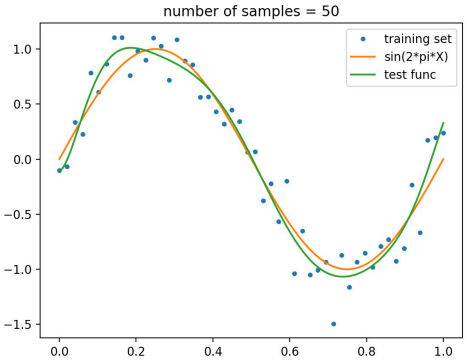
可以观察出，m 在低阶时拟合效果极差，从 m=3 开始较为贴近原函数曲线，一直到 m=5 时有较好的拟合效果。而当 m 再逐渐增大，比如达到 9 阶时我们可以清晰的看到过拟合的现象（此时对应的方程组有唯一解），生成的模式穿过了训练集中所有样本点，模式的复杂度较高，但泛化能力差，但对原函数拟合效果较差。

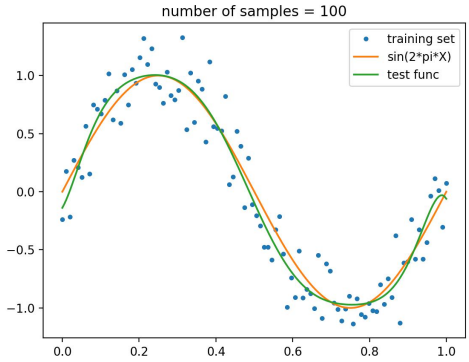
(2).固定阶数，改变样本数

通过上面几张图可以知道 m 在达到一定程度的时候，训练结果会产生过拟合的现象，泛化能力较差而导致对原函数拟合效果不好。我们通过第一种解决过拟合的方法，也就是增加训练样本的数量（分别取 10、30、50、100 个点），以加强模型的泛化能力，改善对原函数的拟合效果：





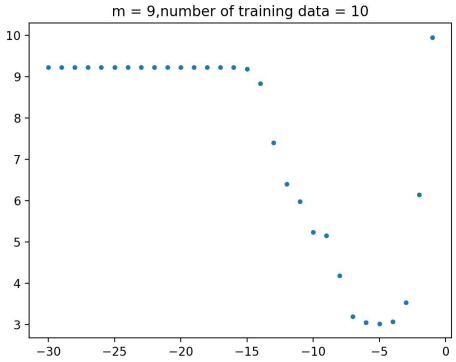


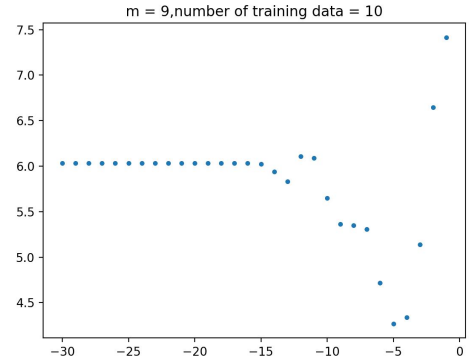


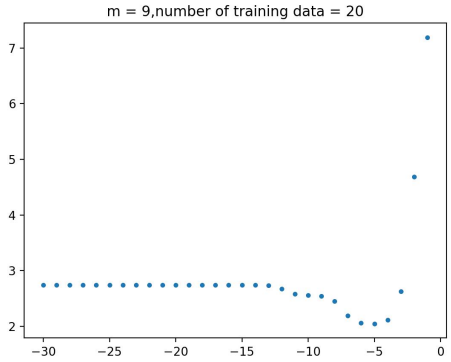
我们可以看到，虽然在只有 10 个样本点时拟合效果很差，但只需要增加一些数据（比如取到 20）就可以一定程度缓解这种状况。在 m=50 时拟合效果已经比较不错，而增加到 100 后则得到一个比较理想、对原函数拟合效果较好的模型。

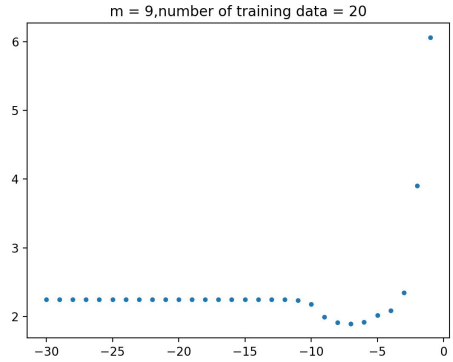
2.解析解（含惩罚项）

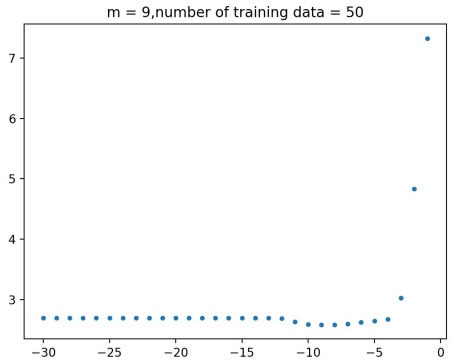
根据课程讲义我们可以知道参数多时，参数向量 **w** 往往有较大的绝对值，这使得用于拟合的曲线不够平滑，因此我们可以加入惩罚项对参数比重产生影响，比重大时降低模型复杂度，比重适当时模型复杂度与问题匹配，比重小时则退化成原模型。这个比重就是我们的超参数 λ，我们需要设置多个λ值确定对我们的数据效果最好的，下面是训练样本数分别为 10、20、50、100 时损失函数值随λ变化的曲线：

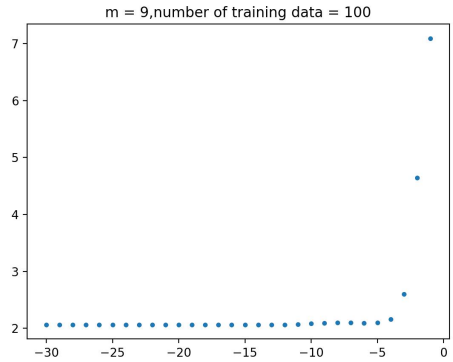










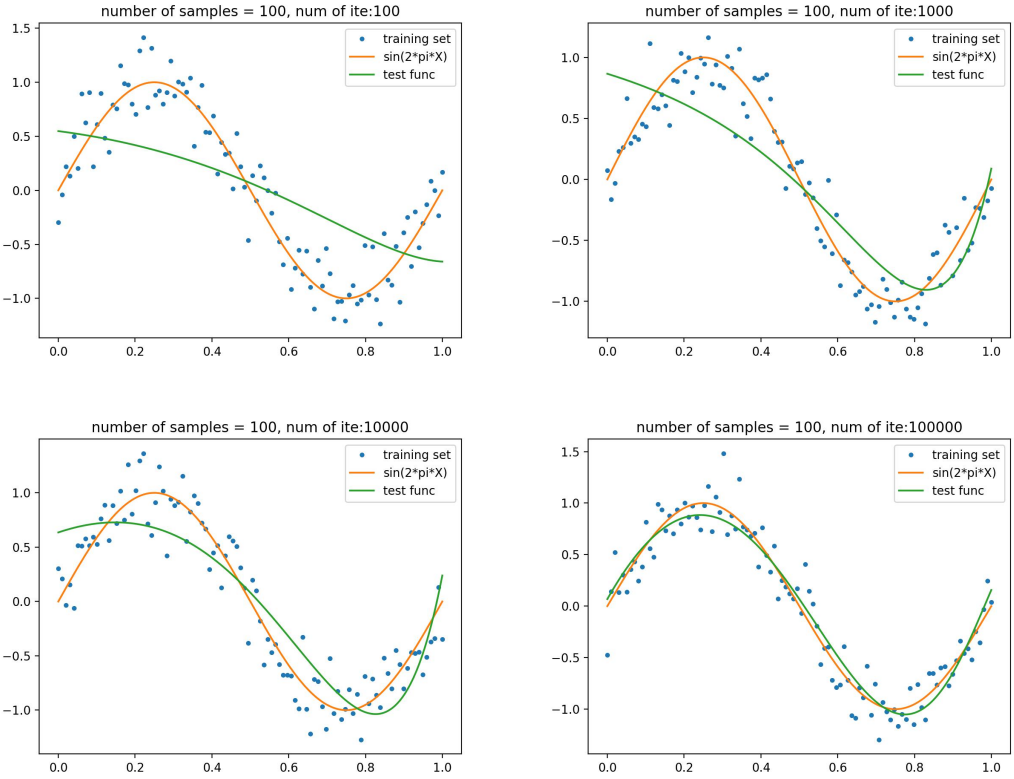


由上图可知，在训练数据量较小时，需要加大惩罚项的权重，λ取到 $1*10^{-5}$ 测试集上的损失最小，较为合适；当数据量适当加大时，取到 $1*10^{-7}$ 左右较为合适。当数据增加到 50 至一百个点，可以取 $1*10^{-6}$ 到 $1*10^{-10}$ 就能有较好的拟合效果。

3.梯度下降法

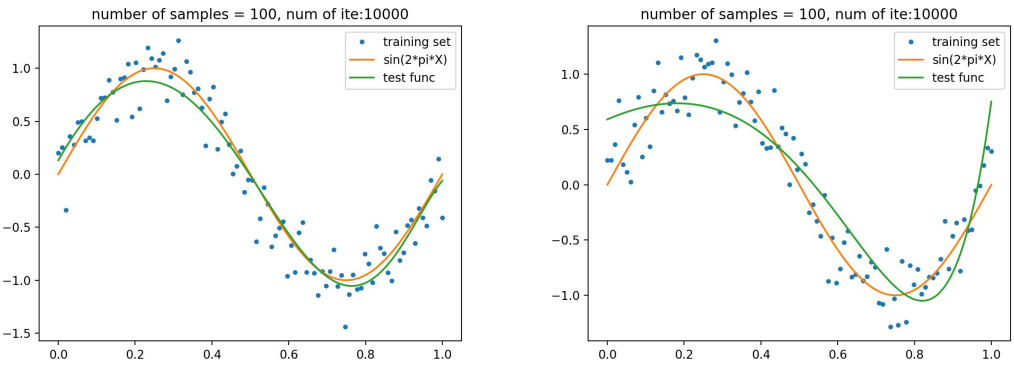
(1).不含惩罚项

这一方法求解问题需要设置学习率、迭代次数等超参数，先来观察学习率相同时(设为 0.001)迭代次数对数据的影响：



可以看到，如果训练样本数和学习率一定，当迭代次数逐渐增大时，模型对原函数的拟合效果变强。因为在学习率合适的情况下，模型关于训练样本的错误率逐渐减少，对原函数的拟合效果逐渐增加。

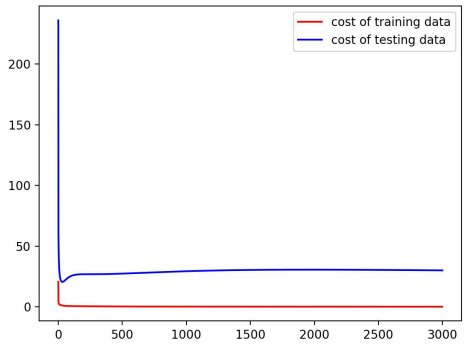
我们再来观察当训练量一定时，不同学习率对拟合效果的影响(下图学习率分别为 0.01 和 0.001)：

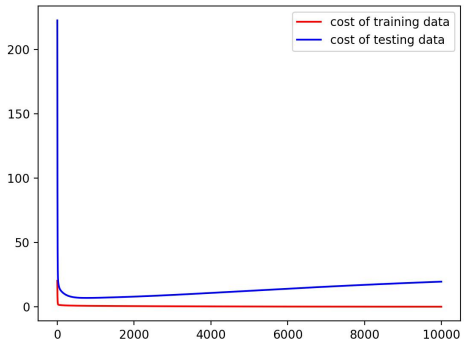


可以观测到，学习率在未使函数发散的范围内减小时，迭代相同次数的拟合效果

变差。当然学习率过大可能导致震荡发散。

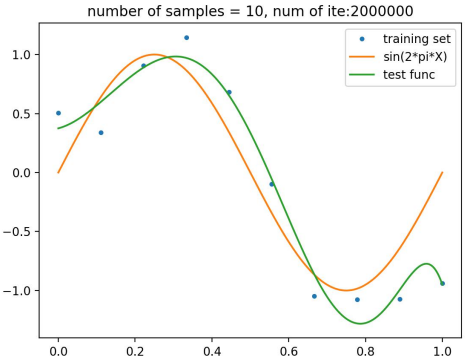
最后，还需要观察在训练样本数极少（便于观察）的情况下训练误差和测试集上的 cost 随迭代次数的变化情况，我们将样本数设为 5，分别以 0.01 的学习率进行 3 万次迭代（左图）、0.001 的学习率进行 100000 次迭代（右图），观察曲线变化情况：

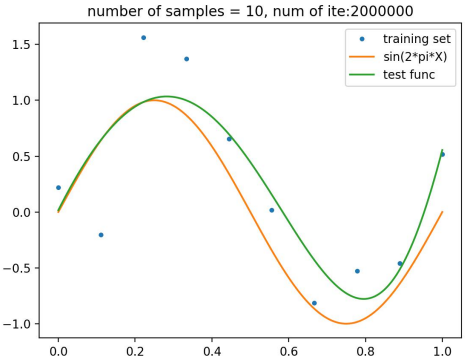




(2).含惩罚项

我们应用前面求得的 lambda 的值的范围，不妨将 lambda 设为 $1*10^{-5}$，观察样本数较少时添加正则项的效果：

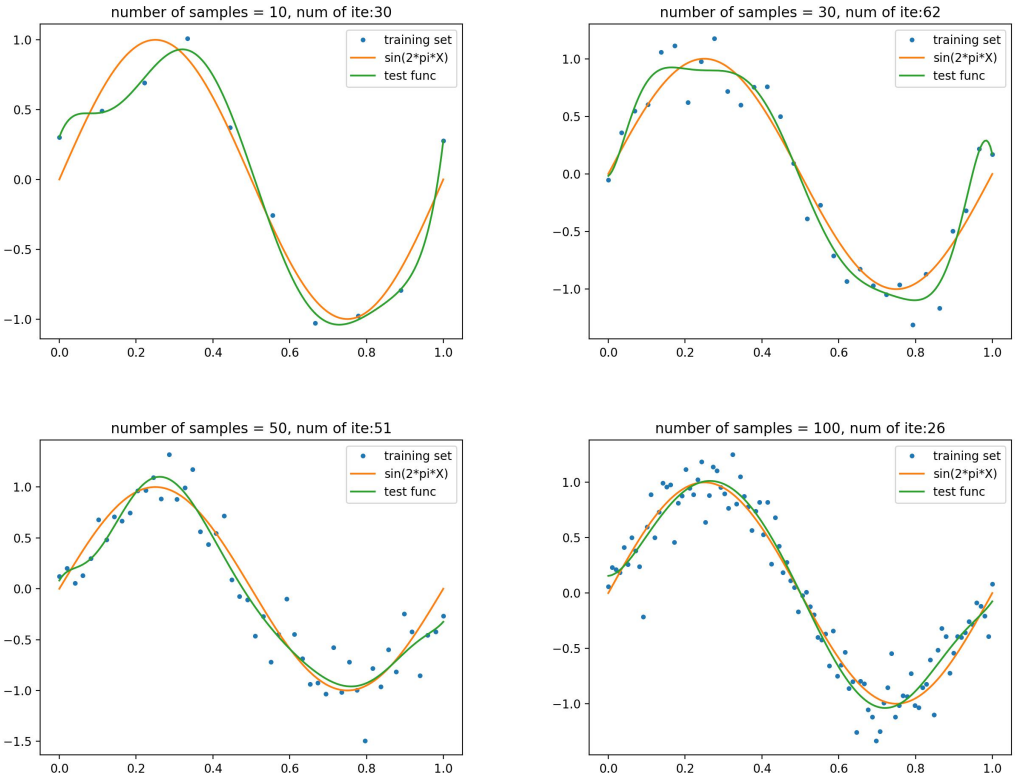




左图为未添加惩罚项的训练结果，右图为添加 lambda 取 $1*10^{-4}$ 次方时的惩罚项得到的结果，可以看到添加惩罚项后曲线更加平滑，模型复杂度得以降低。

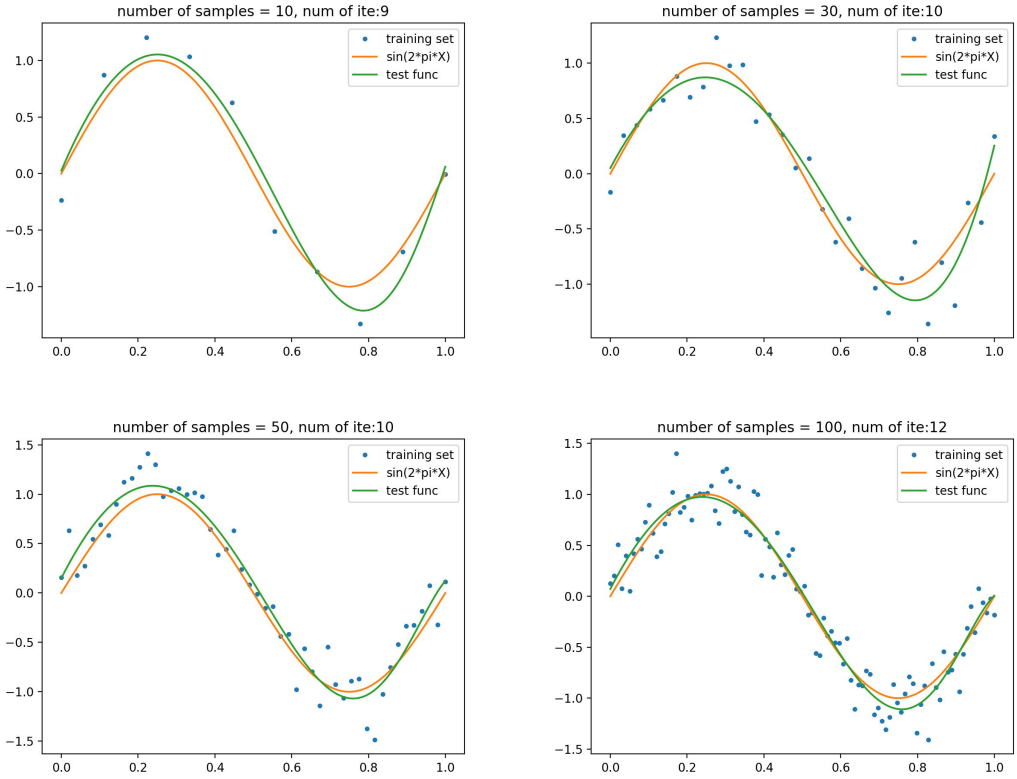
4.共轭梯度法

我们对共轭梯度法与梯度下降法进行迭代次数上的对比，设定训练样本数为 10、30、50、100，m=9,在无正则项时，分别得到如下图像：



可以看到共轭梯度法只需要迭代数十次就可以很好的拟合训练样本，而前面提到的梯度下降法往往需要上万次的迭代。

下面为添加正则项的情况：



可以看到，在训练数据量较少的情况下，加入正则项使得模型对原函数的拟合效果大大增强，拟合出的曲线更加的平滑。另外，迭代次数相对于未加正则项的训练过程有所减少。

五、结论

根据上述实验结果以及算法原理的分析，我们可以得出以下结论：

1.在训练样本较少的情况下，三种算法均容易出现过拟合的情况，此时可以添加正则项降低模型的复杂度，从而平滑曲线优化拟合效果。

2.加大训练样本的数量可以削减过拟合的现象。

3.对于用多项式曲线拟合三角函数而言，当多项式的阶数设置的过小时会出现欠

拟合的结果，而阶数过高时则会出现过拟合的结果，前者对训练和测试集的误差均较大，而后者的泛化能力较差。

4.对于梯度下降法而言，设置合适的学习率（步长）可以加快收敛速度而不至于发散。

5.共轭梯度法的收敛速度（迭代次数）远快于同一问题下的梯度下降法，这与共轭梯度法的算法原理有关。

六、参考资料

吴恩达：机器学习

维基百科：共轭梯度法

七、源代码

main.py:

该部分为 client 端，可以通过调整“控制参数区”中的阶数、样本量来更改模型训练的参量。

```
import numpy as np
import matplotlib.pyplot as plt
import random
import GeneSample #生出数据
```

```
import gradientDecent #梯度下降法
import AnalicalSolution #解析解
import CG #共轭梯度法
import costFunc
import Polycompute


#该方法用于 client 输入字符串选择优化拟合的方法：GD 、CG、解析解
def fit_method(M,X_head,T,str):
    W = np.random.rand(M + 1).reshape(-1,1)   # 随机产生参数向量
    flag = int(input("是否需要在 cost fuction 中加入正则项：'0' 或 '1'"))
    if str == 'GD':
        alpha = float(input("请输入学习率："))
        numOfItem = int(input("请输入迭代次数"))
        if flag == 0 :
            return
gradientDecent.gradient_decent_withoutPun(X_head,T,W,alpha,numOfItem)
        else:
            lambd = float(input('请输入惩罚项系数:'))
            return
gradientDecent.gradient_decent_withPun(X_head,T,W,alpha,numOfItem,la
mbd)
    if str == 'AS':
```

```
            if flag == 0:
                return AnalicalSolution.analical_solution_withoutPun(X_head,T)
            else:
                lambd = float(input('请输入惩罚项系数:'))
                return
AnalicalSolution.analical_solution_withPun(X_head,T,lambd)
        elif str == 'CG':
            if flag == 0:
                return CG.co_gradient_withoutPun(X_head,T,W)
            else:
                lambd = float(input('请输入惩罚项系数:'))
                return CG.co_gradient_withPun(X_head,T,W,lambd)
        else:
            print("没有该方法")
            return


#请见实验报告：实验结果分析部分——2.解析解（含惩罚项）
def choose_lambda_by_AS():


    number_of_tra = 100 #手动设置样本数量 观察 cost 随 labmda 变化的曲
线
    X,T = GeneSample.gene_sample(0,1,number_of_tra)
```

```
    X_head = np.c_[X ** 0, X] #生成样本对应的矩阵 X_head
    for i in range(2, 10):
        X_head = np.c_[X_head, X ** i]
    X_test ,T_test = GeneSample.gene_sample(0,1,100)
    X_h_test = np.c_[X_test ** 0, X_test] #生成测试集对应的矩阵 X_h_head
    for i in range(2, 10):
        X_h_test = np.c_[X_h_test, X_test ** i]

    Lambd = [pow(10,-i) for i in range(30,0,-1)]
    cost = [] #对于每一个 lambda  存储它在测试集上的损失
    for lambd in Lambd:
        W = AnalicalSolution.analical_solution_withPun(X_head,T,lambd)
        Yt = Polycompute.poly_compute(X_h_test,W)
        cost.append(costFunc.cost_func_withoutPun(Yt,T_test))
    print(cost)
    plt.plot(range(-30,0),cost,linestyle = '',marker = '.')
    plt.title('m = 9,number of training data = '+str(number_of_tra))
    plt.show()
    return
# choose_lambda_by_AS() #通过解析解来判断正则项的权重

#————————————————下面区域用于控制阶数、超参数等
```

```
————————————————————————

M =9 #控制阶数

numOfSam = 100 #控制训练样本数


X,T = GeneSample.gene_sample(0,1,numOfSam) #生成训练样本

X_head = np.c_[X**0, X]

for i in range(2, M+1):

    X_head = np.c_[X_head, X ** i]  # 生成样本对应的矩阵 也就是样本 x 中
各点的 0 到 m 次方组成的矩阵


#根据调用 fit_method 方法时最后一个参数的不同选择不同的方法（求解析解、
梯度下降法、共轭梯度法）

# W,cost,test = fit_method(M,X_head,T,'GD')

# W = fit_method(M,X_head,T,'AS')

W,k = fit_method(M,X_head,T,'CG')


plt.plot(X,T,linestyle = '',marker = '.',label = 'training set') #在图像中标注训练
集点

Xfig = np.linspace(0,1,200,endpoint=True).reshape(-1,1)

X_headfig = np.c_[Xfig**0, Xfig]

for i in range(2, M+1):

    X_headfig = np.c_[X_headfig, Xfig ** i] #生成用于描绘训练出的模型的点
```

```
集
plt.plot(Xfig,np.sin(2*np.pi*Xfig),label = 'sin(2*pi*X)') #先画出 sin（2pix）原
曲线
plt.plot(Xfig,np.dot(X_headfig,W),label = 'test func') #再给出拟合的曲线
plt.title('number of samples = '+str(numOfSam))
# plt.plot(cost,color = 'r',label = 'cost of training data') #这两行代码分别用
于展示训练过程中
# plt.plot(test,color = 'b',label = 'cost of testing data') #训练集和测试集的
cost
plt.legend()
plt.show()
```

Polycompute.py:

计算多项式的值

```
import numpy as np
#计算单个样本的值
def poly_compute(X,W):
    Y = np.dot(X,W)
    return Y
```

GeneSample.py

生成训练样本、测试集数据

```
import numpy as np
import random


def gene_sample(sta,end,numOfPoint):
    X = np.linspace(sta,end,numOfPoint,endpoint = True)
    T = np.sin(2*np.pi*X)
    mu = 0 #mu and sigma of gauss distribution
    sigma = 0.2
    for i in range(X.size):
        # X[i] += random.gauss(mu,sigma)
        T[i] += random.gauss(mu,sigma)
    return X.reshape(X.size,1),T.reshape(T.size,1)


def gene_standard_sample(sta,end,numOfPoint):
    X = np.linspace(sta,end,numOfPoint,endpoint = True)
    T = np.sin(2*np.pi*X)
    return X.reshape(X.size,1),T.reshape(T.size,1)
```

AnalyticalSolution.py

解析解

```
import numpy as np


def analical_solution_withoutPun(X,T):
    K = np.linalg.inv(np.dot(X.T,X))
    W = np.dot(np.dot(K,X.T),T)
    return W


def analical_solution_withPun(X,T,lambd):
    K = np.linalg.inv(np.dot(X.T,X)+lambd*np.identity(X.shape[1]))
    W = np.dot(np.dot(K,X.T),T)
    return W
```

gradientDecent.py

梯度下降法

```
import numpy as np
import Polycompute
import costFunc
import GeneSample
#学习率 alpha 待确定
```

```
def gradient_decent_withoutPun(X_head,T,W,alpha,num_ite):
    cost = []
    test = []
    Xt,Tt =GeneSample.gene_standard_sample(0,1,100)
    X_headt = np.c_[Xt ** 0, Xt]
    for i in range(2, len(W)):
        X_headt = np.c_[X_headt, Xt ** i]

    for i in range(num_ite):
        K = np.dot(X_head.T, X_head)
        dW = np.dot(K , W) – np.dot(X_head.T, T)
        W = W – alpha * dW
        if i%10 == 0:
            Y = Polycompute.poly_compute(X_head, W)
            cost.append(costFunc.cost_func_withoutPun(Y,T))
            Yt = Polycompute.poly_compute(X_headt,W)
            test.append(costFunc.cost_func_withoutPun(Yt,Tt))
    return W,cost,test

def gradient_decent_withPun(X_head,T,W,alpha,num_ite,lambd):
    cost = []
    test = []
```

```
    Xt,Tt= GeneSample.gene_standard_sample(0, 1, 100)
    X_headt = np.c_[Xt ** 0, Xt]
    for i in range(2, len(W)):
        X_headt = np.c_[X_headt, Xt ** i]


    for i in range(num_ite):
        # K = np.dot(W.T, X_head)
        # dW = np.sum(X_head * (K – T), axis=1) + lambd * W
        K = np.dot(X_head.T,X_head)
        dW = np.dot(K   ,W) –np.dot(X_head.T,T) + lambd*W
        W = W – alpha * dW
        if i % 10 == 0:
            Y = Polycompute.poly_compute(X_head, W)
            cost.append(costFunc.cost_func_pun(Y, T,lambd,W))
            Yt = Polycompute.poly_compute(X_head, W)
            test.append(costFunc.cost_func_withoutPun(Yt, Tt,lambd,W))
    return W, cost,test
```

CG.py

共轭梯度法：

```
import numpy as np
```

```
def co_gradient_withoutPun(X,T,W):
    A = np.dot(X.T,X) # X (m,n)
    b = np.dot(X.T,T)
    X = W
    r = b - np.dot(A,X)
    p = r
    k = 0
    while True:
        alpha = np.dot(r.T,r)/np.dot(np.dot(p.T,A),p)
        X = X + alpha*p
        k += 1
        r_next = r - alpha* np.dot(A,p)
        if np.linalg.norm(r_next) < 0.0000001:
            break
        beta = np.dot(r_next.T,r_next)/np.dot(r.T,r)
        p = r_next + beta * p
        r = r_next
    print("迭代了"+str(k)+"次")
    return X,k


def co_gradient_withPun(X,T,W,lambd):
    A = np.dot(X.T, X)+lambd*np.identity(X.T.shape[0])
```

```
    b = np.dot(X.T, T)
    X = W #上面三行将参数代入线性方程组
    r = b - np.dot(A, X)
    p = r
    k = 0
    while True:
        alpha = np.dot(r.T, r)/np.dot(np.dot(p.T, A), p) #计算步长
        X = X + alpha * p #对参数 W，也就是这里的 X 进行迭代
        k += 1
        r_next = r - alpha * np.dot(A, p) #计算下一个残差，后面还用到当前的残差，故需要保留
        if np.linalg.norm(r_next) < 0.000001:
            break #当残差小于某个值我们认为拟合效果足够，停止迭代
        beta = np.dot(r_next.T, r_next) / np.dot(r.T, r) #计算搜索方向式子中需要的参数
        p = r_next + beta * p #更新搜索方向
        r = r_next #更新残差
    print("迭代了"+str(k)+"次")
    return X,k
```