

哈尔滨工业大学计算机科学与技术学院

# 实验报告

课程名称：机器学习

课程类型：必修

实验题目：多项式拟合正弦函数

学号：1160300314

姓名：朱明彦

# 一、实验目的

---

掌握最小二乘法求解（无惩罚项的损失函数），掌握增加惩罚项（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（如增加惩罚项、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

- OS: Microsoft Windows 10.0.17134
- Python 3.6.4

## 三、设计思想(本程序中用到的主要算法及数据结构)

---

### 算法原理

#### 1、生成数据算法

主要是利用 $\sin(2\pi x)$ 函数产生样本，其中 $x$ 均匀分布在 $[0, 1]$ 之间，对于每一个目标值 $t = \sin(2\pi x)$ 增加一个0均值，方差为0.5的高斯噪声。

#### 2、利用高阶多项式函数拟合曲线(不带惩罚项)

利用训练集合，对于每个新的 $\hat{x}$ ，预测目标值 $\hat{t}$ 。采用多项式函数进行学习，即利用式(1)来确定参数 $w$ ，假设阶数 $m$ 已知。

$$y(x, w) = w_0 + w_1 x + \cdots + w_m x^m = \sum_{i=0}^m w_i x^i \quad (1)$$

采用最小二乘法，即建立误差函数来测量每个样本点目标值 $t$ 与预测函数 $y(x, w)$ 之间的误差，误差函数即式(2)

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N \{y(x_i, \mathbf{w}) - t_i\}^2 \quad (2)$$

将上式写成矩阵形式如式(3)

$$E(\mathbf{w}) = \frac{1}{2} (\mathbf{X}\mathbf{w} - \mathbf{T})' (\mathbf{X}\mathbf{w} - \mathbf{T}) \quad (3)$$

其中

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & \cdots & x_1^m \\ 1 & x_2 & \cdots & x_2^m \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & \cdots & x_N^m \end{bmatrix}, \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_m \end{bmatrix}, \mathbf{T} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{bmatrix}$$

通过将上式求导我们可以得到式(4)

$$\frac{\partial E}{\partial \mathbf{w}} = \mathbf{X}'\mathbf{X}\mathbf{w} - \mathbf{X}'\mathbf{T} \quad (4)$$

令  $\frac{\partial E}{\partial \mathbf{w}} = 0$  我们有式(5)即为  $\mathbf{w}^*$

$$\mathbf{w}^* = (\mathbf{X}'\mathbf{X})^{-1} \mathbf{X}'\mathbf{T} \quad (5)$$

### 3、带惩罚项的多项式函数拟合曲线

在不带惩罚项的多项式拟合曲线时，在参数多时  $\mathbf{w}^*$  具有较大的绝对值，本质就是发生了过拟合。对于这种过拟合，我们可以通过在优化目标函数式(3)中增加  $\mathbf{w}$  的惩罚项，因此我们得到了式(6)。

$$\tilde{E}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N \{y(x_i, \mathbf{w}) - t_i\}^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2 \quad (6)$$

同样我们可以将式(6)写成矩阵形式，我们得到式(7)

$$\tilde{E}(\mathbf{w}) = \frac{1}{2} [(\mathbf{X}\mathbf{w} - \mathbf{T})' (\mathbf{X}\mathbf{w} - \mathbf{T}) + \lambda \mathbf{w}'\mathbf{w}] \quad (7)$$

对式(7)求导我们得到式(8)

$$\frac{\partial \tilde{E}}{\partial \mathbf{w}} = \mathbf{X}'\mathbf{X}\mathbf{w} - \mathbf{X}'\mathbf{T} + \lambda \mathbf{w} \quad (8)$$

令  $\frac{\partial \tilde{E}}{\partial \mathbf{w}} = 0$  我们得到  $\mathbf{w}^*$  即式(9)，其中  $\mathbf{I}$  为单位阵。

$$\mathbf{w}^* = (\mathbf{X}'\mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}'\mathbf{T} \quad (9)$$

### 4、梯度下降法求解最优解

对于  $f(\mathbf{x})$  如果在  $\mathbf{x}_i$  点可微且有定义，我们知道顺着梯度  $\nabla f(\mathbf{x}_i)$  为增长最快的方向，因此梯度的反方向  $-\nabla f(\mathbf{x}_i)$  即为下降最快的方向。因而如果有式(10)对于  $\alpha > 0$  成立，

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \alpha \nabla f(\mathbf{x}_i) \quad (10)$$

那么对于序列 $\mathbf{x}_0, \mathbf{x}_1, \dots$  我们有

$$f(\mathbf{x}_0) \geq f(\mathbf{x}_1) \geq \dots$$

因此，如果顺利我们可以得到一个  $f(\mathbf{x}_n)$  收敛到期望的最小值，当然对于我们此次实验很大可能性可以收敛到最小值。

## 5、共轭梯度法求解最优解

共轭梯度法解决的主要是形如 $\mathbf{Ax} = \mathbf{b}$ 的线性方程组解的问题，其中 $\mathbf{A}$ 必须是对称的、正定的。求解的方法就是我们先猜一个解 $\mathbf{x}_0$ ，然后取梯度的反方向  $\mathbf{p}_0 = \mathbf{b} - \mathbf{Ax}$ ，在n维空间的基中 $\mathbf{p}_0$ 要与其与的基共轭并且为初始残差。

然后对于第k步的残差  $\mathbf{r}_k = \mathbf{b} - \mathbf{Ax}$ ， $\mathbf{r}_k$ 为 $\mathbf{x} = \mathbf{x}_k$ 时的梯度反方向。由于我们仍然需要保证  $\mathbf{p}_k$  彼此共轭。因此我们通过当前的残差和之前所有的搜索方向来构建 $\mathbf{p}_k$ ，得到式(11)

$$\mathbf{p}_k = \mathbf{r}_k - \sum_{i < k} \frac{\mathbf{p}_i^T \mathbf{Ar}_k}{\mathbf{p}_i^T \mathbf{Ap}_i} \mathbf{p}_i \quad (11)$$

进而通过当前的搜索方向 $\mathbf{p}_k$ 得到下一步优化解 $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ ，其中

$$\alpha_k = \frac{\mathbf{p}_k^T (\mathbf{b} - \mathbf{Ax}_k)}{\mathbf{p}_k^T \mathbf{Ap}_k} = \frac{\mathbf{p}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{Ap}_k}$$

## 算法的实现

对于数据生成、求解析解（有无正则项）都是可以利用numpy中的矩阵求逆等库变相降低了算法实现的难度，因此在这里就不再赘述，下面主要讲梯度下降法和共轭梯度法的算法实现。

### 1、梯度下降

此处我们利用带惩罚项的优化函数进行梯度下降法的实现。由梯度下降主要解决的是如式(12)的线性方程组的解。

$$\mathbf{Ax} - \mathbf{b} = 0 \quad (12)$$

另由式(8)我们可以得到式(13)

$$J(\mathbf{w}) = (\mathbf{X}'\mathbf{X} + \lambda\mathbf{I})\mathbf{w} - \mathbf{X}'\mathbf{T} \quad (13)$$

联合式(13)与式(12)我们可以有：

$$\begin{cases} \mathbf{A} = \mathbf{X}'\mathbf{X} + \lambda\mathbf{I} \\ \mathbf{b} = \mathbf{X}'\mathbf{T} \end{cases} \quad (14)$$

进而我们实现算法如下，其中 $\delta$ 为精度要求，通常可以设置为 $\delta = 1 \times 10^{-6}$ ：

```

rate = 0.01, k = 0
w0 = 0
 $\tilde{E}(\mathbf{w}) = \frac{1}{2N}[(\mathbf{X}\mathbf{w} - \mathbf{T})'(\mathbf{X}\mathbf{w} - \mathbf{T}) + \lambda\mathbf{w}'\mathbf{w}]$ 
 $J(\mathbf{w}) = (\mathbf{X}'\mathbf{X} + \lambda\mathbf{I})\mathbf{w} - \mathbf{X}'\mathbf{T}$ 
loss0 =  $\tilde{E}(\mathbf{w}_0)$ 
repeat :
    wk+1 = wk - rate * J(wk)
    lossk+1 =  $\tilde{E}(\mathbf{w}_{k+1})$ 
    if abs(lossk+1 - lossk) < δ then break loop
    k = k + 1
end repeat

```

## 2、共轭梯度下降

对于共轭梯度下降，算法实现如下，参考[wiki](#)实现：

```

r0 := b - Ax0
p0 := r0
k := 0
repeat
     $\alpha_k := \frac{\mathbf{r}_k^T \mathbf{r}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}$ 
    xk+1 := xk + αkpk
    rk+1 := rk - αkA pk
    if rk+1 is sufficiently small, then exit loop
     $\beta_k := \frac{\mathbf{r}_{k+1}^T \mathbf{r}_{k+1}}{\mathbf{r}_k^T \mathbf{r}_k}$ 
    pk+1 := rk+1 + βkpk
    k := k + 1
end repeat
The result is xk+1

```

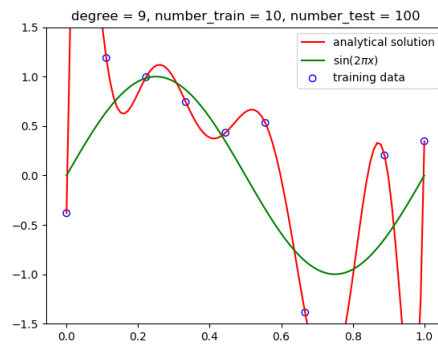
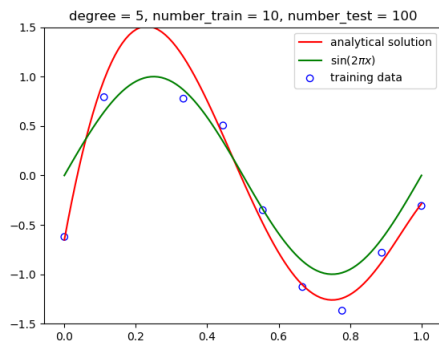
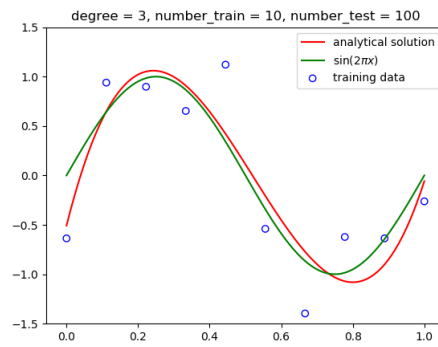
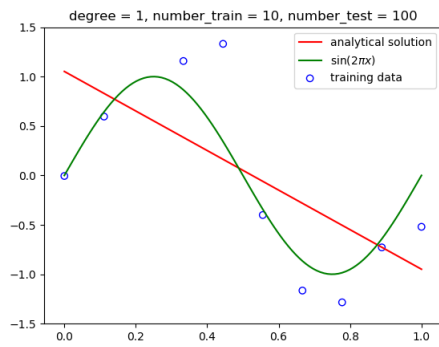
其中的b, A均与式(14)相同。

## 四、实验结果分析

---

### 1、不带惩罚项的解析解

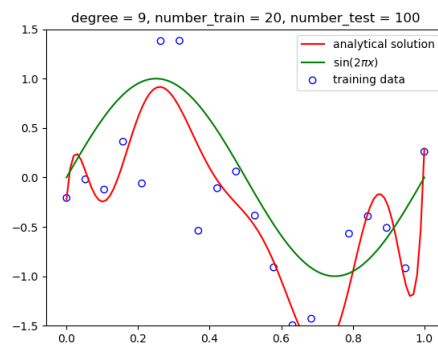
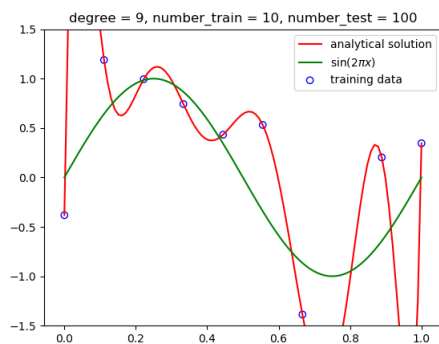
(1)固定训练样本的大小为10，分别使用不同多项式阶数，测试的结果如下图。

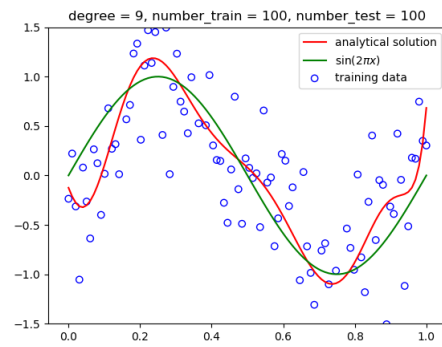
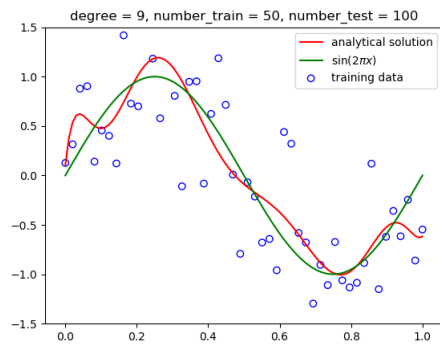


我们可以看到在固定训练样本的大小之后，在多项式阶数为3时的拟合效果已经很好。继续提高多项式的阶数，尤其在阶数为9的时候曲线“完美的”经过了所有的节点，这种剧烈的震荡并没有很好的拟合真实的背后的函数 $\sin(2\pi x)$ ，反而将所有噪声均很好的拟合，即表现出来一种过拟合的情况。其出现过拟合的本质原因是，在阶数过大的情况下，模型的复杂度和拟合的能力都增强，因此可以通过过大或者过小的系数来实现震荡以拟合所有的数据点，以至于甚至拟合了所有的噪声。在这里由于我们的数据样本大小只有10，所以在阶数为9的时候，其对应的系数向量 $\mathbf{w}$ 恰好有唯一解，因此可以穿过所有的样本点。

对于过拟合我们可以通过增加样本的数据或者通过增加惩罚项的方式来解决。增加数据集样本数量，使其超过参数向量的大小，就会在一定程度上解决过拟合问题。

(2)固定多项式阶数，使用不同数量的样本数据，测试的结果如下图。





我们可以看到在固定多项式阶数为9的情况下，随着样本数量逐渐增加，过拟合的现象有所解决。特别是对比左上图与右下图的差距，可以看到样本数量对于过拟合问题是有影响的。

## 五、结论

---

## 六、参考文献

---

- [Pattern Recognition and Machine Learning](#).
- [Gradient descent wiki](#)
- [Conjugate gradient method wiki](#)

## 七、附录:源代码(带注释)

---

```
print("Hello world")

import numpy as np

np.one(10)
```