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：选修

实验题目： 多项式拟合正弦函数

学号：1171800323
姓名： 杨富祥

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：Windows10 + matlabR2018a

三、设计思想（本程序中的用到的主要算法及数据结构）

1．算法原理

输入：给定 m 个样本$\left(\mathrm{x}^{(1)},\mathrm{y}^{(1)}\right),\left(\mathrm{x}^{(2)},\mathrm{y}^{(2)}\right),\dots\left(\mathrm{x}^{(\mathrm{m})},\mathrm{y}^{(\mathrm{m})}\right)$，每个$\mathrm{x}^{(\mathrm{i})}=\left[\mathrm{x}_0^{(\mathrm{i})},\mathrm{x}_1^{(\mathrm{i})},\mathrm{x}_2^{(\mathrm{i})},\dots,\mathrm{x}_\mathrm{n}^{(\mathrm{i})}\right]^\mathrm{T}$都有 n 个特征，其中$\mathrm{x}_0^{(i)}=1$。

目标：定义拟合函数为$\mathrm{h}_\theta(x)=\theta_0+\theta_1x_1+\cdots+\theta_nx_n=\theta^\mathrm{T}x$.为了让

拟合函数更加接近原函数，用最小二乘法定义代价函数$\mathrm{J}(\theta) = \frac{1}{2\mathrm{m}}\Sigma_{\mathrm{i=1}}^{\mathrm{m}}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$，并求$\min \mathrm{J}(\theta)$。

输出：θ向量。

2. 算法的实现

1）梯度下降法

从代价函数的任意一点θ出发,每次沿着当前位置下滑最快的方向走,也就是该点处的梯度方向，就可以得到代价函数最小值。从而得到一系列的解序列:$\theta^{(1)}, \theta^{(2)}, \cdots$直到两次下降的差小于给定的误差限为止。

给定初始值θ，然后每次沿梯度方向更新：$\theta_{\mathrm{j}} \coloneqq \theta_j - \frac{\alpha\partial}{\partial\theta_j}J(\theta)$.即可得到如下算法：

Repeat until convergence{

$$\theta_{\mathrm{j}} \coloneqq \theta_j - \alpha\Sigma_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)} \quad (for\ every\ j).$$

}

向量化计算可以调用现有函数，避免写过多循环，会更简单，可化为：

while true{

for i=1 to m,{

$$\delta := \delta + \left(\theta^T x^{(i)} - y^{(i)}\right)x^{(i)}$$

}

$$\delta := \frac{\delta}{\mathrm{m}}$$

$$\theta \coloneqq \theta - \alpha\delta$$

if J(θ)在θ更新前后之差$< 10^{-8}$

break

}

2）共轭梯度法

最速下降法每一次的迭代过程,下降的方向都与上一次的方向正交.在最速下降法中,有一个很糟糕的现象,为了收敛到解附近,同样的迭代方向可能走了不止一次。每一次的方向都是特征向量的线性组合,而且大多数情况下,前一次迭代过的特征向量方向上的分量,在下一次的迭代中还继续存在。

选取一系列线性无关的方向向量,沿着每个方向只走一次,最后就能到达解处。

每一次迭代之间的残差都是相互正交的，选残差作为共轭化之前的基。由于使用共轭化的方向向量来迭代至多只有 n 步，且每步都将该方向上的误差消灭掉，故而这一组基不仅线性无关，而且由于它们是残差，还具有正交的良好性质。

下面是共轭梯度法涉及到的所有公式：

$$
\begin{aligned}
d_{(0)} &= r_{(0)} = b - Ax_{(0)} \\
\alpha_{(i)} &= \frac{r_i^T r_{(i)}}{d_{(i)}^T A d_{(i)}} \\
x_{(i+1)} &= x_{(i)} + \alpha_{(i)} d_{(i)} \\
r_{(i+1)} &= r_{(i)} - \alpha_{(i)} A d_{(i)} \\
\beta_{(i+1)} &= \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}} \\
d_{(i+1)} &= r_{(i+1)} + \beta_{(i+1)} d_{(i)}
\end{aligned}
$$

具体实现来说，令$\nabla_\theta J(\theta)=0$，可得方程：$X^T X\theta = X^T y$. 并且$X^T X$是对称正定阵，可以采用共轭梯度法。

```
function x = CG(X,y)

A = X' * X;
b = X' * y;

x = zeros(size(A,1),1);
r = b - A * x;
p = r;

for k = 1 : size(A,1)
    if r == 0
        break;
    end
    alpha = (r' * r) / ((A * p)' * p);
    x = x + alpha * p;
    temp = r' * r;
    r = r - alpha * A * p;
    belta = r' * r / temp;
    p = r + belta * p;
end
```

3）正规方程法（解析解）

$$\begin{aligned}\frac{1}{2}(X\theta-\vec{y})^T(X\theta-\vec{y}) &= \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2 \\ &= J(\theta)\end{aligned}$$

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \frac{1}{2}(X\theta-\vec{y})^T(X\theta-\vec{y}) \\ &= \frac{1}{2}\nabla_\theta\left(\theta^T X^T X\theta-\theta^T X^T\vec{y}-\vec{y}^T X\theta+\vec{y}^T\vec{y}\right) \\ &= \frac{1}{2}\nabla_\theta \operatorname{tr}\left(\theta^T X^T X\theta-\theta^T X^T\vec{y}-\vec{y}^T X\theta+\vec{y}^T\vec{y}\right) \\ &= \frac{1}{2}\nabla_\theta\left(\operatorname{tr}\theta^T X^T X\theta-2\operatorname{tr}\vec{y}^T X\theta\right) \\ &= \frac{1}{2}\left(X^T X\theta+X^T X\theta-2X^T\vec{y}\right) \\ &= X^T X\theta-X^T\vec{y}\end{aligned}$$

$$X^T X\theta = X^T\vec{y}$$

$$\theta = (X^T X)^{-1}X^T\vec{y}.$$

4）正则化

如果数据量很少，但是拟合函数的阶数很高，会产生过拟合现象。为了让模型更简单，令每一个θ_j更小，从而达到更好的拟合效果。加入对θ的惩罚：

$$J(\theta)=\frac{1}{2m}\left(\Sigma_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2+\lambda\Sigma_{j=1}^{n}\theta_j^2\right)。$$

在实现时，不对θ_0惩罚。

正规方程的解变为：

$$\theta=\left(X^TX+\lambda\begin{bmatrix}0 & \cdots & 0\\ \vdots & \ddots & \vdots\\ 0 & \cdots & 1\end{bmatrix}\right)^{-1}X^Ty.$$ (对角线除第一个是 0，其余都是 1)。

梯度下降法变为：

Repeat {

$$\theta_0 := \theta_0-\alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)})-y^{(i)}\right)x_0^{(i)}$$

$$\theta_j := \theta_j-\alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)})-y^{(i)}\right)x_j^{(i)}$$

$$(j=\cancel{0},\underline{1,2,3,\ldots,n})$$

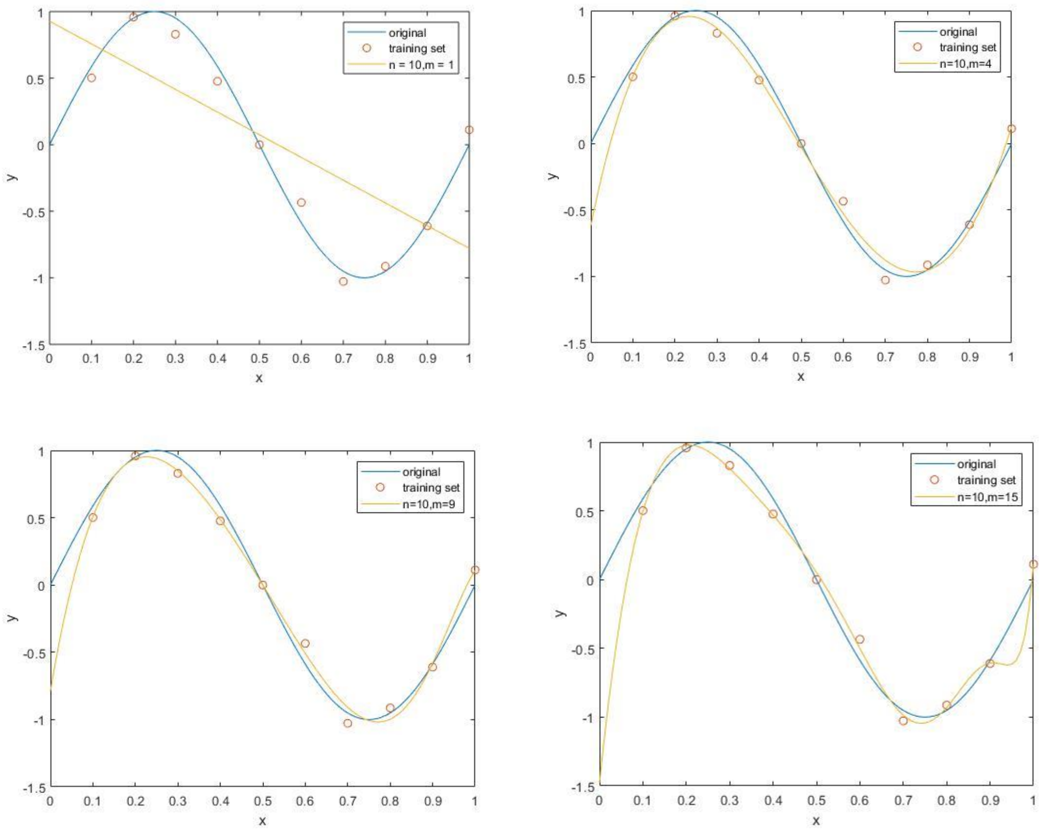
}

四、实验结果与分析

1．N=10 时，增加多项式函数的阶数 M，在 M=15 出现过拟合

M=1	M=4	M=9	M=15
0.4133	-0.6265	-0.787	-1.4798
-0.7714	15.5363	18.6036	30.6722
	-45.9843	-63.6394	-127.239
	38.0286	71.4829	183.988
	-6.8253	-4.6611	-7.8005
		-46.4057	-135.847
		-9.8842	-77.5188

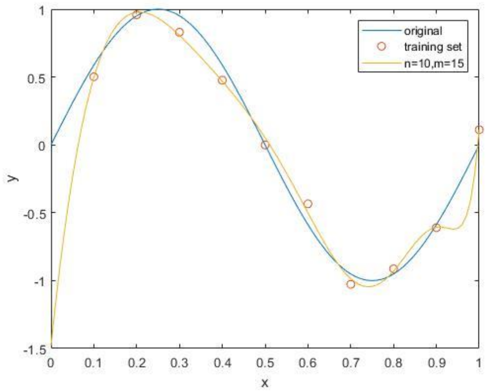
		37.2134	42.7197
		35.2401	114.2355
		-37.0544	103.6871
			33.2825
			-52.7106
			-110.659
			-108.184
			-26.7715
			139.7366

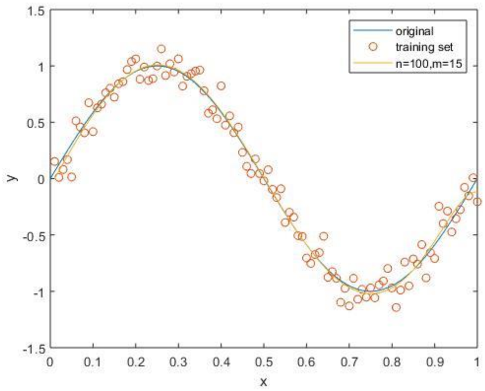


2. M=15，训练样本数量增加可以减少过学习程度

N=10	N=100
-1.4798	-0.0691
30.6722	6.8116
-127.239	0.583
183.988	-55.2525
-7.8005	45.8619
-135.847	26.2784
-77.5188	-6.4925
42.7197	-15.3431
114.2355	-9.4314
103.6871	-1.8608

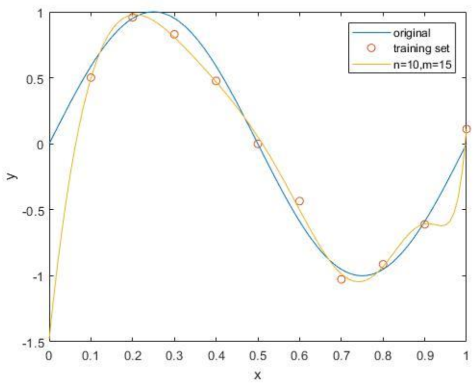
33.2825	2.2606
-52.7106	3.6893
-110.659	4.4476
-108.184	4.7892
-26.7715	2.1563
139.7366	-8.5463

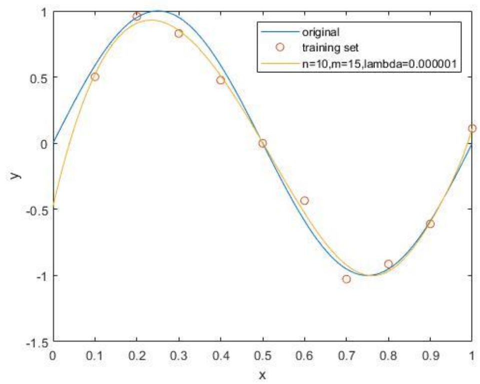


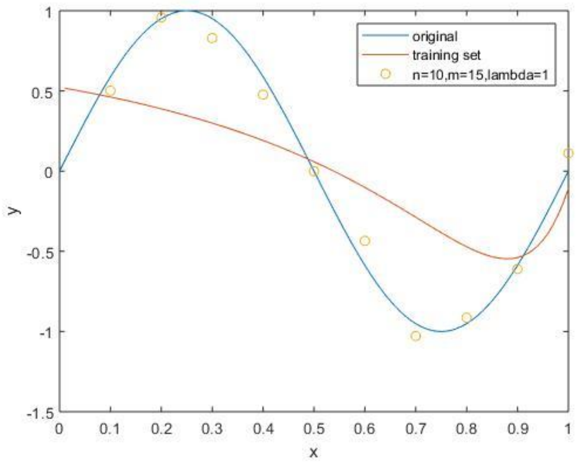


3. M=15，N=10 时，加入正则项后的实验

$ln\lambda = -\infty$	$ln\lambda = -13.8155$	$ln\lambda = 0$
-1.4798	-0.4769	0.5224
30.6722	13.6127	-0.5544
-127.239	-39.1176	-0.5018
183.988	27.2375	-0.3547
-7.8005	8.5765	-0.2202
-135.847	-13.2682	-0.1149
-77.5188	-10.767	-0.0359
42.7197	3.0549	0.0229
114.2355	12.9356	0.0668
103.6871	12.9837	0.0997
33.2825	5.0281	0.1247
-52.7106	-5.6164	0.1437
-110.659	-13.2976	0.1585
-108.184	-13.6207	0.1699
-26.7715	-3.9351	0.179
139.7366	16.781	0.1861







五、结论

1．给定样本数量较少时，多项式函数的阶数过大，会出现过拟合现象；阶数太小，模型表达能力有限，会造成欠拟合；阶数为 4-9 时，拟合效果较好。

2. 训练样本数量增加可以减少过拟合程度。

3. 加入正则项可以降低模型的复杂程度，减少过拟合程度。λ选取，如果太大，只有θ_0一个参数会起作用，太小惩罚效果不太好，需要仔细选择。

4. 梯度下降法步长选取，要自己尝试，可以依次选择 0.001，0.003，0.1，0.3，1……过大会导致无法收敛，损失函数下降后增大，过小则迭代次数太多，时间太长。

5. CG 法可以在 n 次迭代内收敛，速度极快。

6. 正规方程求解析解时，求$(X^T X)^{-1}$，当矩阵阶数太大时，可能会花费许多时间。

六、参考文献

1. https://alkane0050.fun/2019/05/18/%E5%85%B1%E8%BD%AD%E6%A2%AF%E5%BA%A6%E6%B3%95%E5%88%9D%E6%AD%A5/?tdsourcetag=s_pctim_aiomsg

2. https://see.stanford.edu/materials/aimlcs229/cs229-notes1.pdf

七、附录：源代码（带注释）

```
function [X,x,z] = productData(N,Order)

% N 为产生数据的个数
% Order 为多项式的阶数

% X 为数据矩阵，N * (Order + 1)
% x 为 N 个数据的横坐标向量
% z 为增加高斯噪声后的纵坐标向量

X = ones(N,1);
gap = 1 / N;
x = gap : gap : 1;
for i = 1 : Order
```

```
   temp = [X x(:).^i];
   X = temp;
end
y = (sin(2 * pi * x))';
z = y + 0.1*randn(N,1);
function theta = gradientDescent(X,y,theta,alpha)

% X 为数据矩阵
% y 为给定函数值向量
% theta 为函数参数
% alpha 为步长

delta = zeros(size(X,2),1);
m = size(X,1);
cost = 0;
number=0;
while true
  for i = 1 : m
    delta = delta + (theta' * X(i,:)' - y(i)) * (X(i,:)
');
  end
  delta = delta / m;
```

```
  theta = theta - alpha * delta;    % 更新 theta

  number = number + 1;              % 记录迭代次数


  predictions = X * theta;

  sqrErrors = (predictions-y).^2;


  J = 1 / (2 * m) * sum(sqrErrors);

  if abs(J-cost) < 1e-12   % 代价函数减小的量小于一定值
后，结束循环

      disp(number);

      break;

  end

  cost = J;

end
```

```
function theta = normalEquation(X,y)


theta = pinv(X'*X)*X'*y;
```

```
function x = CG(X,y)
```

```
A = X' * X; % 构造对称正定矩阵
b = X' * y;

x = zeros(size(A,1),1);
r = b - A * x;
p = r;

 for k = 1 : size(A,1)
     if r == 0
        break;
    end
    alpha = (r' * r) / ((A * p)' * p);
    x = x + alpha * p;
    temp = r' * r;
    r = r - alpha * A * p;
    belta = r' * r / temp;
    p = r + belta * p;
 end
```

```
function theta = normalEquationRegularization(X,y,lambd
a)
```

```
% lambda 为惩罚系数

columns = size(X,2);
temp = eye(columns);
temp(1,1) = 0;

theta = pinv(X' * X - lambda * temp) * X' * y;
```

```
function x = CGRegularization(X,y,lambda)

columns = size(X,2);
temp = eye(columns);
temp(1,1) = 0;

A = X' * X + lambda * temp;
b = X' * y;

x = zeros(size(A,1),1);
r = b - A * x;
p = r;
```

```
for k = 1 : size(A,1)
    if r == 0
       break;
   end
   alpha = (r' * r) / ((A * p)' * p);
   x = x + alpha * p;
   temp = r' * r;
   r = r - alpha * A * p;
   belta = r' * r / temp;
   p = r + belta * p;
end
```

```
function theta = gradientDescentRegularization(X,y,thet
a,alpha,lambda)

% X 为数据矩阵
% y 为给定函数值向量
% theta 为函数参数
% alpha 为步长
% lambda 为惩罚系数
```

```
delta = zeros(size(X,2),1);

m = size(X,1);

cost = 0;

number=0;

while true

  for i = 1 : m

    delta = delta + (theta' * X(i,:)' - y(i)) * (X(i,:)
');

  end

  delta = delta / m;

  temp = theta(1,1);

  theta = theta * (1 - alpha * lambda / m) - alpha * de
lta;

  theta(1,1) = temp - alpha * delta(1,1);

  number = number + 1;


  predictions = X * theta;

  sqrErrors = (predictions-y).^2;


  J = 1 / (2 * m) * sum(sqrErrors);

  if abs(J-cost) < 1e-12

      disp(number);
```

```
      break;
  end
  cost = J;
end
```