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：1173710204

姓名： 陈东鑫

1. 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)

1. 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：Windows10

语言：python3

编程环境：jupyter notebook

1. 设计思想（本程序中的用到的主要算法及数据结构）
2. 算法原理
3. 生成数据

在正弦函数的一个周期中均匀产生m个点，并加入随机噪声，即

其中，sound是使用python的随机数生成的，（，）代表第i个数据

1. 假设空间

使用多项式

去拟合数据

函数的实现代码

1. #多项式函数
2. **def** H(W,X):
3. H = np.dot(W.T,X)
4. **return** H[0][0]
5. 损失函数

函数的实现代码：

1. #平方损失函数
2. **def** loss(x,y,W,M,N):
3. loss = 0
4. **for** i **in** range(N):
5. X = np.ones((M+1,1))
6. **for** j **in** range(M+1):
7. X[j][0] = X[j][0]\*(x[i]\*\*j)
8. loss = (H(W,X) - y[i])\*\*2 + loss
9. loss = loss/(2\*N)
10. **return** loss
11. 梯度下降

损失函数的梯度

其中grad为一个n行列向量

实现代码

1. #梯度
2. **def** grad(x,y,W,M,N):
3. grad = np.ones((M+1,1))
4. sum = 0
5. **for** k **in** range(M+1):
6. **for** i **in** range(N):
7. X = np.ones((M+1,1))
8. **for** j **in** range(M+1):
9. X[j][0] = x[i]\*\*j
10. sum = sum + (H(W,X) - y[i])\*x[i]\*\*k
11. grad[k][0] = sum/N
12. **return** grad

梯度下降法

1. 最小二乘法

公式

代码实现

1. temp1 = np.dot(X.T,X)
2. temp2 = np.linalg.inv(temp1)
3. temp3 = np.dot(temp2,X.T)
4. W2 = np.dot(temp3,y.reshape((N,1)))
5. 带L2正则项的最小二乘法

公式

其中I是一个n\*n（n是特征数）的单位矩阵

代码实现

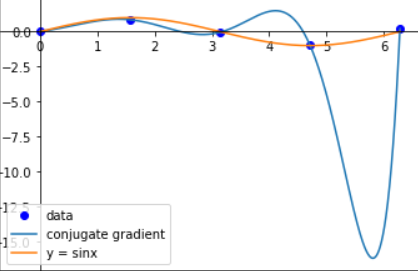
1. temp1 = np.dot(X.T,X) + lam\*N\*I
2. temp2 = np.linalg.inv(temp1)
3. temp3 = np.dot(temp2,X.T)
4. W3 = np.dot(temp3,y.reshape((N,1)))
5. 共轭梯度

代码实现

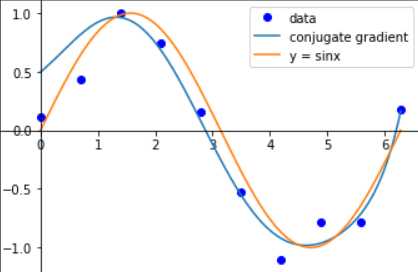
1. #共轭梯度法
2. #A是取样点x形成的矩阵
3. #b是y
4. #x0是参数w形成的矩阵
5. **def** conGrad(A, b, x0):
6. b = b.reshape((b.size,1))
7. x = x0
8. p = b - np.dot(A,x0)
9. r = p
11. **for** k **in** range(x0.size):
12. **if** (r==np.zeros\_like(r)).all():
13. **print**("shutdown in r")
14. **break**
15. alpha = ((np.dot(r.T,r))/np.dot(np.dot(A,p).T,p))[0][0]
16. **print**("alpha:\n",alpha)
17. x = x + alpha\*p
18. temp = r
19. r = r - alpha\*np.dot(A,p)
20. belta = np.dot(r.T,r)[0][0]/np.dot(temp.T,temp)[0][0]
21. p = r+belta\*p
22. **return** x
23. 算法的实现
24. #梯度下降法
25. M1 = 3
26. a = 0.0001
27. W1 = param(M1)
28. oldLoss = 0
29. newLoss = loss(x,y,W1,M1,N)
30. times = 16000
31. **for** i **in** range(times):
32. W1 = W1 - a\*grad(x,y,W1,M1,N)
33. oldLoss = newLoss
34. newLoss = loss(x,y,W1,M1,N)
35. t = abs(newLoss-oldLoss)
36. **print**("第",i+1,"次迭代，平方损失函数为：",newLoss,"，与上次相差：",t)
37. **print**("各项参数为：\n",W1)
38. **if** t<1e-5:
39. **break**
40. **if** newLoss > 1e+16:
41. **break**
43. #不加正则项的解析解
44. M2 = 3
45. W2 = param(M2)
47. X = np.ones((N,M2+1))
48. **for** i **in** range(N):
49. **for** j **in** range(M2+1):
50. X[i][j] = X[i][j]\*(x[i]\*\*j)
52. temp1 = np.dot(X.T,X)
53. temp2 = np.linalg.inv(temp1)
54. temp3 = np.dot(temp2,X.T)
55. W2 = np.dot(temp3,y.reshape((N,1)))
57. #加L2正则项的解析解
58. M3 = 3
59. W3 = param(M3)
60. I = np.eye(M3+1)
61. lam = 0.001
62. X = np.ones((N,M3+1))
63. **for** i **in** range(N):
64. **for** j **in** range(M3+1):
65. X[i][j] = X[i][j]\*(x[i]\*\*j)
67. temp1 = np.dot(X.T,X) + lam\*N\*I
68. temp2 = np.linalg.inv(temp1)
69. temp3 = np.dot(temp2,X.T)
70. W3 = np.dot(temp3,y.reshape((N,1)))
71. **print**(temp1)
72. **print**(temp2)
73. **print**(temp3)
74. **print**(W3)
76. #共轭梯度法
77. M4 = 5
78. W4 = param(M4)
79. X = np.ones((N,M4+1))
80. **for** i **in** range(N):
81. **for** j **in** range(M4+1):
82. X[i][j] = X[i][j]\*(x[i]\*\*j)
83. b = np.dot(X.T,y)
84. X = np.dot(X.T,X)
85. W4 = conGrad(X,b,W4)
86. 实验结果与分析
87. 样本数对结果的影响

统一使用8次多项式拟合

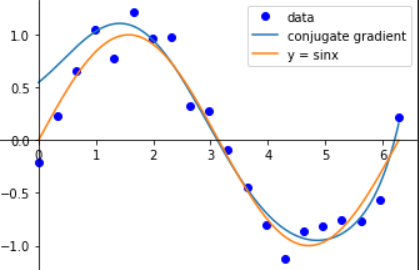
5个样本



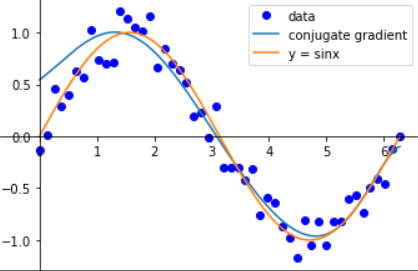
1. 10个样本



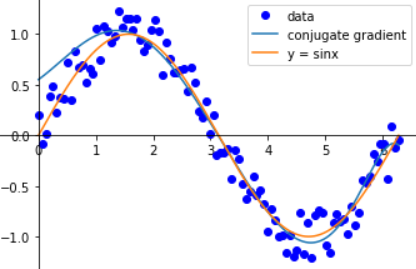
1. 20个样本



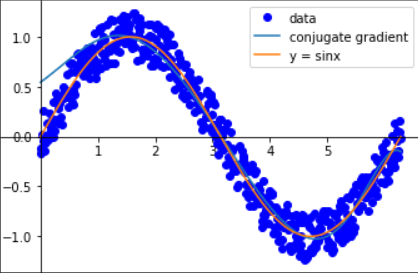
1. 50个样本



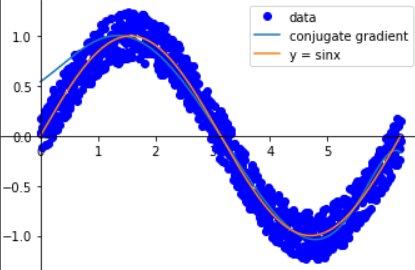
1. 100个样本



1. 500个样本

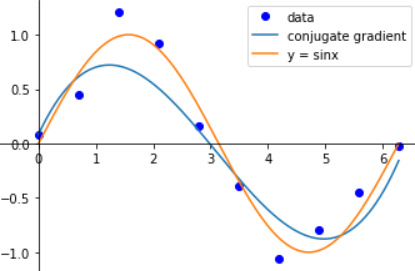


1. 1000个样本

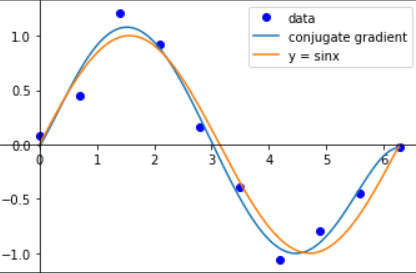


1. 特征数对结果的影响

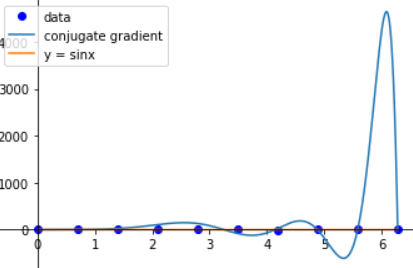
3次多项式，10个样本



5次多项式



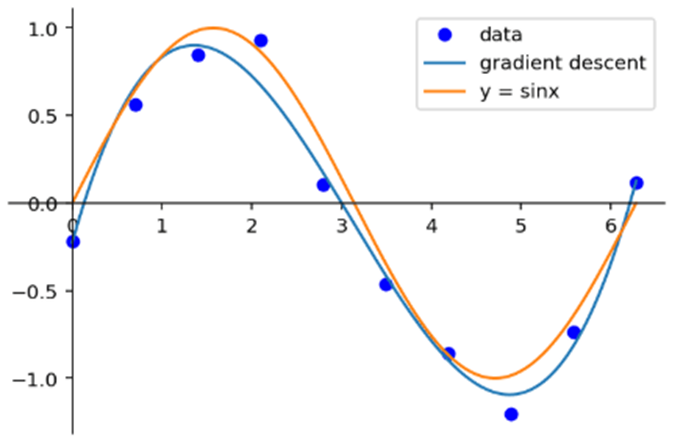
14次多项式



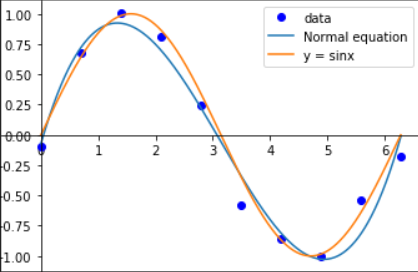
1. 拟合方法对结果的影响

十个样本点，三次函数进行拟合

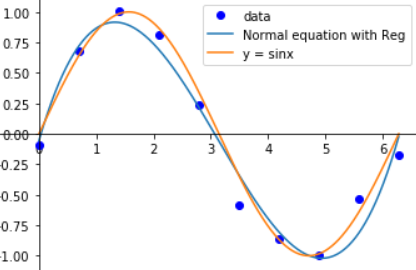
梯度下降法



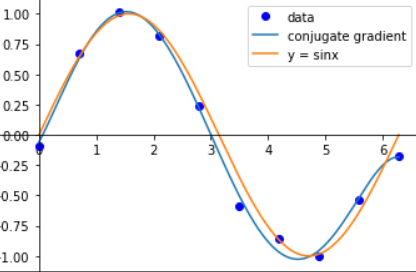
最小二乘法



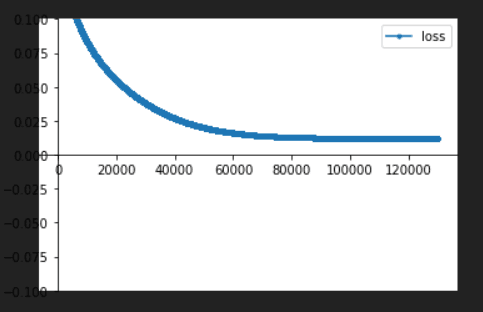
带L2正则项的最小二乘法



共轭梯度法



三次多项式拟合的时候的loss画图如下，找不到比较好的角度展示这个图，只好这样了（注意纵坐标是loss值，取值范围是-0.1到0.1），横坐标是迭代次数



1. 结论

可以看出，当样本数低于用于拟合的多项式的次数时，会出现过拟合现象。这一现象可以通过增大样本数来避免。并且梯度下降法需要多次迭代同时收敛缓慢，最小二乘法无需迭代，直接计算即可，共轭梯度法在迭代不超过特征数时就可收敛，得到较为理想的结果

1. 参考文献
2. 附录：源代码（带注释）
3. **import** numpy as np
4. **import** pandas as pd
5. **import** matplotlib.pyplot as plt
6. **import** math
8. #取样函数
9. **def** sample(N):
10. x = np.linspace(0,2\*math.pi,N)
11. y = []
12. **for** i **in** x:
13. noise = (np.random.rand()-0.5)\*0.5
14. y.append(math.sin(i)+noise)
15. x = np.array(x)
16. y = np.array(y)
17. **return** x,y
19. #多项式函数
20. **def** H(W,X):
21. H = np.dot(W.T,X)
22. **return** H[0][0]
24. #产生参数W
25. #M代表次数
26. **def** param(M):
27. W = np.ones((M+1,1))
28. **return** W
30. #平方损失函数
31. **def** loss(x,y,W,M,N):
32. loss = 0
33. **for** i **in** range(N):
34. X = np.ones((M+1,1))
35. **for** j **in** range(M+1):
36. X[j][0] = X[j][0]\*(x[i]\*\*j)
37. loss = (H(W,X) - y[i])\*\*2 + loss
38. loss = loss/(2\*N)
39. **return** loss
41. #梯度
42. **def** grad(x,y,W,M,N):
43. grad = np.ones((M+1,1))
44. sum = 0
45. **for** k **in** range(M+1):
46. **for** i **in** range(N):
47. X = np.ones((M+1,1))
48. **for** j **in** range(M+1):
49. X[j][0] = x[i]\*\*j
50. sum = sum + (H(W,X) - y[i])\*x[i]\*\*k
51. grad[k][0] = sum/N
52. **return** grad
54. #共轭梯度法
55. #A是取样点x形成的矩阵
56. #b是y
57. #x0是参数w形成的矩阵
58. **def** conGrad(A, b, x0):
59. b = b.reshape((b.size,1))
60. x = x0
61. p = b - np.dot(A,x0)
62. r = p
64. **for** k **in** range(x0.size):
65. **if** (r==np.zeros\_like(r)).all():
66. **break**
67. alpha = ((np.dot(r.T,r))/np.dot(np.dot(A,p).T,p))[0][0]
68. x = x + alpha\*p
69. temp = r
70. r = r - alpha\*np.dot(A,p)
71. belta = np.dot(r.T,r)[0][0]/np.dot(temp.T,temp)[0][0]
72. p = r+belta\*p
73. **return** x
75. #x,y是采样点的值，x1，y1是原正弦函数的图形
76. xs = []
77. ys = []
79. #N是取样点的个数
80. N = 10
82. x,y = sample(N)
83. xs = np.linspace(0,2\*math.pi,10000)
84. **for** i **in** xs:
85. ys.append(math.sin(i))
87. #M是多项式的次数
88. #a是学习率
89. a = 0.0001
90. M1 = 3
91. W1 = param(M1)
92. newLoss = loss(x,y,W1,M1,N)
94. epoch = 5000
95. loss\_x = []
96. loss\_y = []
97. **for** i **in** range(epoch):
98. W1 = W1 - a\*grad(x,y,W1,M1,N)
99. oldLoss = newLoss
100. newLoss = loss(x,y,W1,M1,N)
101. loss\_x.append(i+1)
102. loss\_y.append(newLoss)
103. t = abs(newLoss-oldLoss)
104. **print**("第",i+1,"次迭代，平方损失函数为：",newLoss,"，与上次相差：",t)
105. **print**("各项参数为：\n",W1)
106. **if** t<1e-9:
107. **print**("break complete")
108. **break**
109. **if** newLoss > 1e+16:
110. **print**("break too big")
111. **break**
113. #不加正则项的解析解
114. M2 = 3
115. W2 = param(M2)
117. X = np.ones((N,M2+1))
118. **for** i **in** range(N):
119. **for** j **in** range(M2+1):
120. X[i][j] = X[i][j]\*(x[i]\*\*j)
122. temp1 = np.dot(X.T,X)
123. temp2 = np.linalg.inv(temp1)
124. temp3 = np.dot(temp2,X.T)
125. W2 = np.dot(temp3,y.reshape((N,1)))
127. #加L2正则项的解析解
128. M3 = 3
129. W3 = param(M3)
130. I = np.eye(M3+1)
131. lam = 0.001
132. X = np.ones((N,M3+1))
133. **for** i **in** range(N):
134. **for** j **in** range(M3+1):
135. X[i][j] = X[i][j]\*(x[i]\*\*j)
137. temp1 = np.dot(X.T,X) + lam\*N\*I
138. temp2 = np.linalg.inv(temp1)
139. temp3 = np.dot(temp2,X.T)
140. W3 = np.dot(temp3,y.reshape((N,1)))
142. #共轭梯度法
143. M4 = 14
144. W4 = param(M4)
145. X = np.ones((N,M4+1))
146. **for** i **in** range(N):
147. **for** j **in** range(M4+1):
148. X[i][j] = X[i][j]\*(x[i]\*\*j)
149. b = np.dot(X.T,y)
150. X = np.dot(X.T,X)
151. W4 = conGrad(X,b,W4)
153. plt.figure()
154. x1 = np.array(np.linspace(0,2\*math.pi,1000))
155. y1 = []
156. x2 = np.array(np.linspace(0,2\*math.pi,1000))
157. y2 = []
158. x3 = np.array(np.linspace(0,2\*math.pi,1000))
159. y3 = []
160. x4 = np.array(np.linspace(0,2\*math.pi,1000))
161. y4 = []

164. **for** i **in** range(1000):
165. X1 = np.ones((M1+1,1))
166. **for** j **in** range(M1+1):
167. X1[j][0] = X1[j][0]\*(x1[i]\*\*j)
168. y1.append(H(W1,X1))
169. y1 = np.array(y1)
171. **for** i **in** range(1000):
172. X2 = np.ones((M2+1,1))
173. **for** j **in** range(M2+1):
174. X2[j][0] = X2[j][0]\*(x2[i]\*\*j)
175. y2.append(H(W2,X2))
176. y2 = np.array(y2)
178. **for** i **in** range(1000):
179. X3 = np.ones((M3+1,1))
180. **for** j **in** range(M3+1):
181. X3[j][0] = X3[j][0]\*(x3[i]\*\*j)
182. y3.append(H(W3,X3))
183. y3 = np.array(y3)
185. **for** i **in** range(1000):
186. X4 = np.ones((M4+1,1))
187. **for** j **in** range(M4+1):
188. X4[j][0] = X4[j][0]\*(x4[i]\*\*j)
189. y4.append(H(W4,X4))
190. y4 = np.array(y4)
192. plt.plot(x,y,"ob",label='data')
193. plt.plot(x1,y1,label='gradient descent')
194. plt.plot(x2,y2,label='Normal equation')
195. plt.plot(x3,y3,label='Normal equation with Reg')
196. plt.plot(x4,y4,label='conjugate gradient')
197. # plt.plot(xL,yT,label='taile')
198. plt.plot(xs,ys,label='y = sinx')
199. plt.xlim((-0.7, 6.6))
200. # plt.ylim((-2, 2))
202. ax = plt.gca()
203. ax.spines['right'].set\_color('none')
204. ax.spines['top'].set\_color('none')
205. ax.xaxis.set\_ticks\_position('bottom')
206. ax.spines['bottom'].set\_position(('data',0))
207. ax.yaxis.set\_ticks\_position('left')
208. ax.spines['left'].set\_position(('data',0))
210. plt.legend()
211. plt.show()