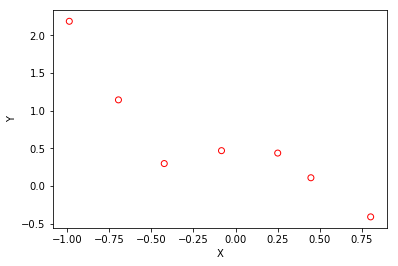
**Programming Assignment 2**

**Part A:**

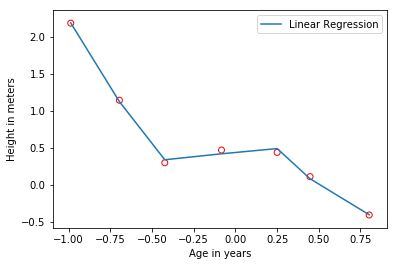
Plot of original data:



lambda: 0

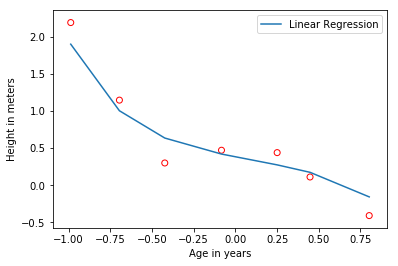
theta: array([ 0.49917479, 0.83306615, -2.03845857, -6.93424088, 3.39424083,
5.75306609])

L2Norm: 9.8894299563366523

lambda: 1

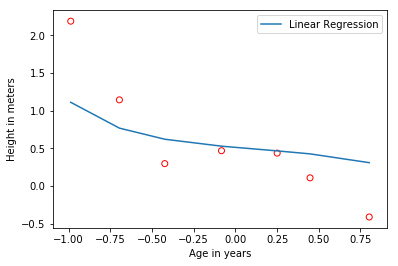
theta: array([ 0.38106374, -0.43781896, 0.12880159, -0.43609185, 0.1933533 ,
-0.37688109])

L2Norm: 0.85034452774484648

lambda: 10

theta: array([ 0.51357264, -0.19100471, 0.06437106, -0.15400044, 0.07903144,
-0.13137918])

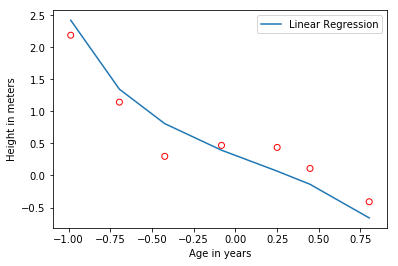
L2Norm: 0.59296363841252353



lambda: -1

theta: array([ 0.31089401, -0.98611001, 0.18116587, -0.48964457, 0.16984902,
-0.34777078])

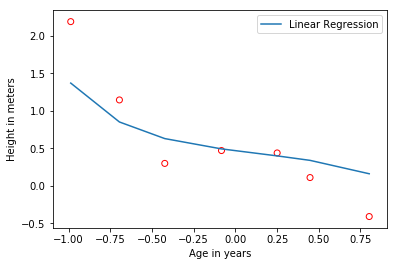
L2Norm: 1.2212428600852612



lambda: 5
t

heta: array([ 0.46756365, -0.28432857, 0.09426909, -0.23548116, 0.1184942 ,
-0.20138849])

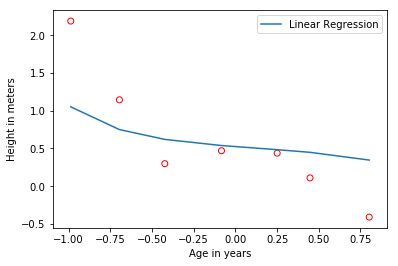
L2Norm: 0.64683439678612598



lambda: 12

theta: array([ 0.52443515, -0.16870046, 0.05701219, -0.13539704, 0.06972039,
-0.1154572 ])

L2Norm: 0.58588916953416492



Discuss how well you expect the theta to generalize to other data points in this distribution: I expect that the thetas that are closer to 0 will generalize better to other data points in this distribution. It matches to the current data points more closely, and I think will also match closer to points generated by the same distribution.

As lambda grows, the line that generalizes to the points gets less and less complex. It becomes almost linear as lambda goes to infinity. It draws a line that correlates most closely to the points with a degree of variability that relates to lambda. Lambda determines how far the line can be from the points it is correlating.

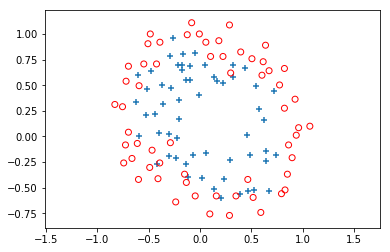
I think a lambda of a value that is very close to 0 will generalize to data the best. This could lead to over-training on our dataset, but I think a value of greater than 5 will not generalize well enough.

Code:

1. #!/usr/bin/env python2
2. # -\*- coding: utf-8 -\*-
3. """
4. Created on Mon Sep 18 20:32:55 2017
6. @author: rditljtd
7. """
9. import numpy as np
10. from mpl\_toolkits.mplot3d import Axes3D
11. import matplotlib.pyplot as plt
12. from map\_features import \*
14. x = np.loadtxt('ax.dat')
15. y = np.loadtxt('ay.dat')
17. #Plot original data out
18. plt.scatter(x, y, facecolors='none', color='red')
19. plt.xlabel('X')
20. plt.ylabel('Y')
21. plt.show()
23. #Get the number of examples
24. m = x.shape[0]
25. #Reshape x to be a 2D column vector
26. x.shape = (m,1)
27. #Add a column of ones to x
28. X = np.hstack([np.ones((m,1)), x, x\*\*2, x\*\*3, x\*\*4, x\*\*5])
30. theta = np.zeros(shape=(6,1))
32. lambdaa = [0, 1, 10, -1, 5, 12]
34. #L2norm = np.linalg.norm(X)
36. #print L2norm
38. **for** i in range(len(lambdaa)):
40. matrix = np.diag([0, 1,1,1,1,1])
41. #print matrix
43. theta = np.linalg.inv(X.transpose().dot(X) + lambdaa[i]\*(matrix)).dot(X.transpose().dot(y))
44. print "lambda: " + `lambdaa[i]`
45. print "theta: " + `theta`
46. print "L2Norm: " + `np.linalg.norm(theta)`
47. #Plot original data out
48. plt.plot(X[:,1],np.dot(X,theta))
49. plt.legend(['Linear Regression', 'Training Data'])
50. plt.scatter(x, y, facecolors='none', color='red')
51. plt.xlabel('Age in years')
52. plt.ylabel('Height in meters')
53. plt.show()
54. i = i+1
56. #def computeCost(X, y, theta):
57. #    m=y.size
58. #    estimates = (X-y[:,None]).dot(theta).flatten()
59. #    squared\_errors = (estimates) \*\* 2
60. #    sum\_errors = (1 / (2 \* m))\*squared\_errors.sum()
61. #    J = lambdaa\*(theta)
62. #    return J
63. #
64. #    #sum\_first = ((hypothesis(X)-y)\*\*2).sum()
65. #    #sum\_second = lambdaa\*((theta\*\*2).sum)
66. #    #J\_theta = (1/(2\*m))\*(sum\_first+sum\_second)
67. #
68. #def gradientDescent(X, y, theta, lambdaa, iterations):
69. #    m = y.size
70. #    J\_history = np.zeros(shape = (iterations, 1))
71. #
72. #    for i in range(iterations):
73. #        J\_history[i,0] = computeCost(X, y, theta)
74. #
75. #    return theta, J\_history
76. #
77. ##theta, J\_history = gradientDescent(X, y, theta, lambdaa, 10)
78. #
79. #print theta

**Part B:**

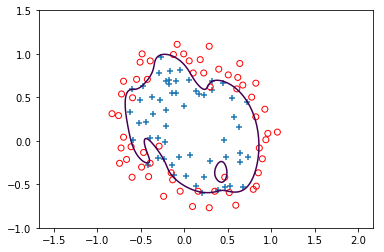
Plot of Raw Data:



Final theta: [[ 27.04247528]
[ 19.82536158]
[ 68.94141094]
...,
[-2670.0856728 ]
[-1307.90297298]
[ -522.34199678]]

Lambda: 0

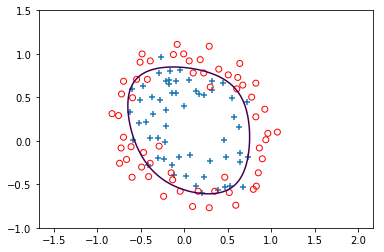
L2Norm: 7301.09758994



Final theta: [[ 1.18455196]
[ 0.68338187]
[ 1.17726894]
...,
[-0.33930754]
[-0.13867704]
[-0.92708495]]

Lambda: 1

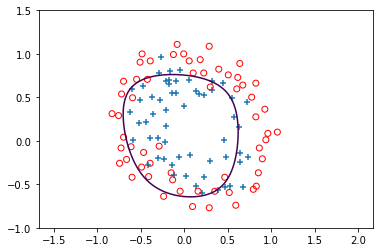
L2Norm: 4.03969126766



Final theta: [[ 0.22865286]
[ 0.00972808]
[ 0.17478993]
...,
[-0.06220219]
[-0.01216701]
[-0.2639623 ]]

Lambda: 10

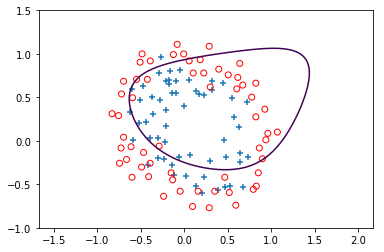
L2Norm: 0.84144167998



Final theta: [[-1.79501504]
[-3.1821499 ]
[-6.62131099]
...,
[ 0.45324879]
[-0.57963481]
[ 3.5393426 ]]

Lambda: -2

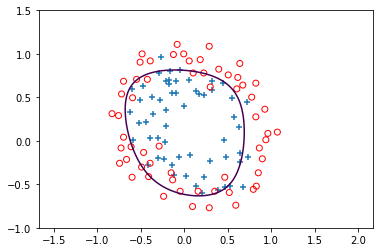
L2Norm: 10.2832842969



Final theta: [[ 0.4987114 ]
[ 0.15580862]
[ 0.43977339]
...,
[-0.13027842]
[-0.0314288 ]
[-0.47722152]]

Lambda: 4

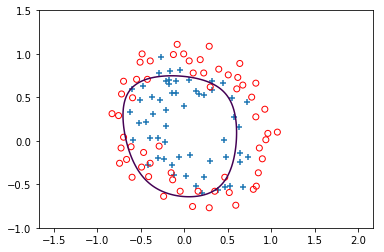
L2Norm: 1.68197469019



Final theta: [[ 0.19064156]
[-0.0036435 ]
[ 0.14142199]
...,
[-0.05331595]
[-0.01022133]
[-0.23178081]]

Lambda: 12

L2Norm: 0.727484211912



I used 15 iterations for all of these plots.

Higher lambda values make the circle that determines the category of the data value more lenient on where it is fit for the data. It is much more strictly fit to the data values that it has been given when the lambda is lower.

I believe that a lambda value of 1 is a fairly close fit without over-fitting to the training data. I believe that a lambda value of 0 is too closely fit and probably over-fit.

Code:

1. #!/usr/bin/env python2
2. # -\*- coding: utf-8 -\*-
3. """
4. Created on Mon Sep 21 20:23:00 2017
6. @author: rditljtd
7. """
9. import numpy as np
10. from mpl\_toolkits.mplot3d import Axes3D
11. import matplotlib.pyplot as plt
12. from scipy.special import expit
13. from map\_features import \*
15. x = np.loadtxt('bx.dat', delimiter=",")
16. y = np.loadtxt('by.dat')

19. ##Get the number of examples
20. #m = x.shape[0]
21. #
22. ##Reshape x to be a 2D column vector
23. #x.shape = (m,2)
24. #
25. ##Add a column of ones to x
26. #X = np.hstack([np.ones((m,1)), x])
27. #
28. ##initialize theta
29. theta = np.zeros(shape=(28,1)) #Initialize theta
30. ##print theta
31. ##alpha = .3#Your learning rate#
32. ##J = []
33. #iterations = 15
34. #J\_history = np.zeros(shape = (iterations, 1))
36. #pos = np.nonzero(y)
37. #neg = np.where(y==0)[0]
38. #
39. #plt.scatter(x[pos,0],x[pos,1],marker='+')
40. #plt.scatter(x[neg,0],x[neg,1],facecolors='none',marker='o', color='r')
41. #plt.show()
43. #Find indices of positive and negative examples
44. pos = np.nonzero(y)
45. neg = np.where(y==0)[0]
46. #Plot out the data
47. plt.scatter(x[pos,0],x[pos,1],marker='+')
48. plt.scatter(x[neg,0],x[neg,1],facecolors='none',marker='o',color='r')
49. plt.axis('equal')
50. plt.show()
52. X=map\_features(x[:,0], x[:,1])
53. #Newtons method solution
55. def sigmoid(z):
56. # print "z: " + `z`
57. toreturn = expit(z)
58. # print "sigmoid: " + `toreturn`
59. **return** toreturn
61. def hypothesis(X, theta):
62. # print "here: " + `theta`
63. # print "there: " + `X`
64. toreturn = (sigmoid(np.dot(X, theta)))
65. # print "hypothesis: " + `toreturn`
66. **return** toreturn
68. def computeCost(X, y, theta, lambdaa):
69. m = y.size
70. firstOp = ((1.0/m) \* (-y\*np.log(hypothesis(X, theta))) - (1.0-y)\*(np.log(1.0-hypothesis(X, theta)))).sum()
71. toreturn = ((1.0\*lambdaa/(2.0\*m)) \* (theta\*\*2.0)).sum() - firstOp
72. #print "computeCost: " + `toreturn`
73. **return** toreturn
75. def gradientDescent(X, y, theta, lambdaa):
76. m = y.size
77. minus = hypothesis(X, theta) - y
78. result = (1.0/m)\*(X.T.dot(minus))
79. toreturn = ((lambdaa\*1.0)/m) \* theta + (result)
80. #print "gradientDescent: " + `toreturn`
81. #print "result: " + `result`
82. **return** toreturn
84. def hessian(X, y, theta, lambdaa):
85. m = y.size
86. minus = (1.0-hypothesis(X,theta))\*(hypothesis(X, theta))
87. Xm = minus\*X
88. result = (1.0/m)\*(Xm.T.dot(X))
89. matrix = np.diag([0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1])
90. toreturn = ((lambdaa\*1.0)/m) \* matrix + (result)
91. # print "hessian: " + `toreturn`
92. #result and toreturn are the same?
93. **return** toreturn
95. def newtonsMethod(X, y, theta, lambdaa, iterations):
96. m=y.size
97. y.shape = (m,1)
98. **for** i in range(iterations):
99. J\_history[i] = computeCost(X, y, theta, lambdaa)
100. theta = (theta - np.linalg.inv((hessian(X, y, theta, lambdaa))).dot(gradientDescent(X, y, theta, lambdaa)))
101. #print theta, J\_history
102. **return** theta, J\_history
104. def runNewtonsMethod(l):
105. lambdaa = l
106. theta1, J\_history = newtonsMethod(X, y, theta, lambdaa, 15)
107. print 'Final theta: ', theta1
108. print 'Lambda: ' + `lambdaa` + ' L2Norm: ', np.linalg.norm(theta1)
110. #Define the ranges of the grid
111. u = np.linspace(-1,1.5,200)
112. v = np.linspace(-1,1.5,200)
113. #Reshape to be 2-D
114. u.shape = (len(u),1)
115. v.shape = (len(v),1)
116. #Plotting commands below
117. X1, Y = np.meshgrid(u,v)
118. Z = np.zeros((len(u),len(v)))
119. #Initialize z = theta\*x over the whole grid
120. **for** i in range(len(u)):
121. **for** j in range(len(v)):
122. Z[j][i] = np.dot(map\_features(u[i],v[j]),theta1)
123. plt.clf()
124. plt.scatter(x[pos,0],x[pos,1],marker='+')
125. plt.scatter(x[neg,0],x[neg,1],facecolors='none',marker='o',color='r')
126. plt.axis('equal')
127. plt.contour(X1,Y,Z, 0,linewidth=2)
128. plt.show()
130. i=0
131. lambdaas=[0, 1, 10, -2, 4, 12]
132. **for** i in range(len(lambdaas)):
133. runNewtonsMethod(lambdaas[i])