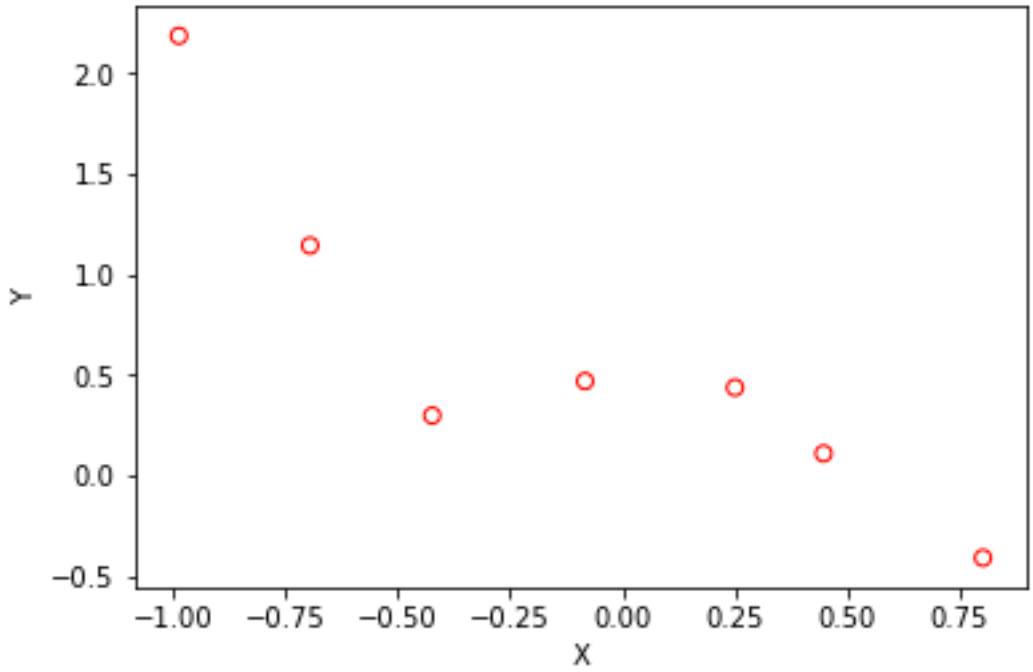
Programming Assignment 2

Part A:

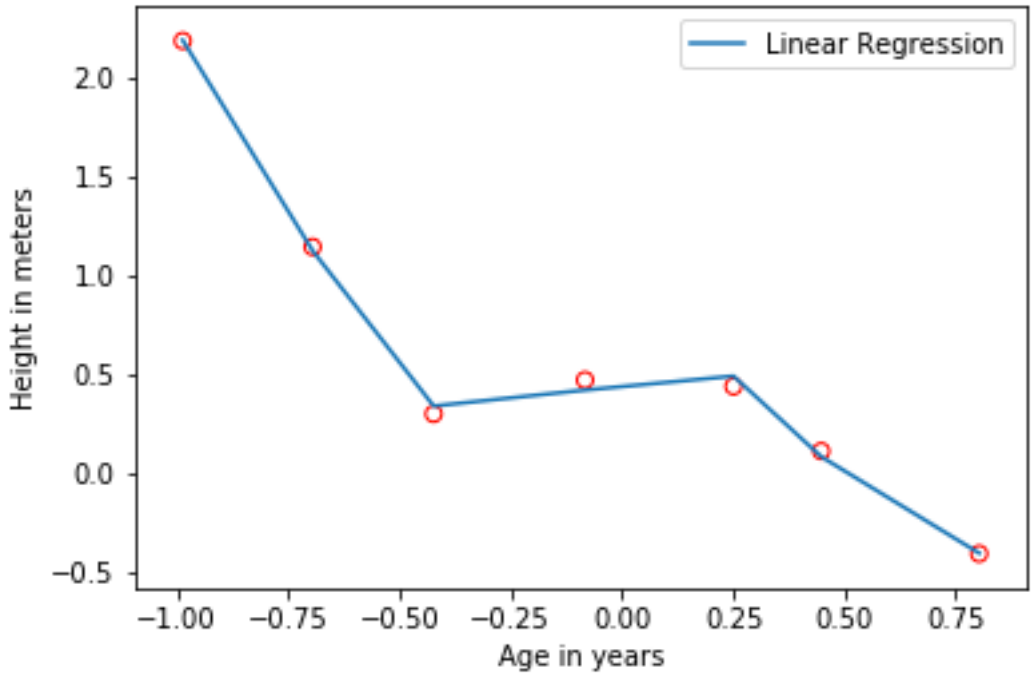
Plot of original data:



lambda: 0

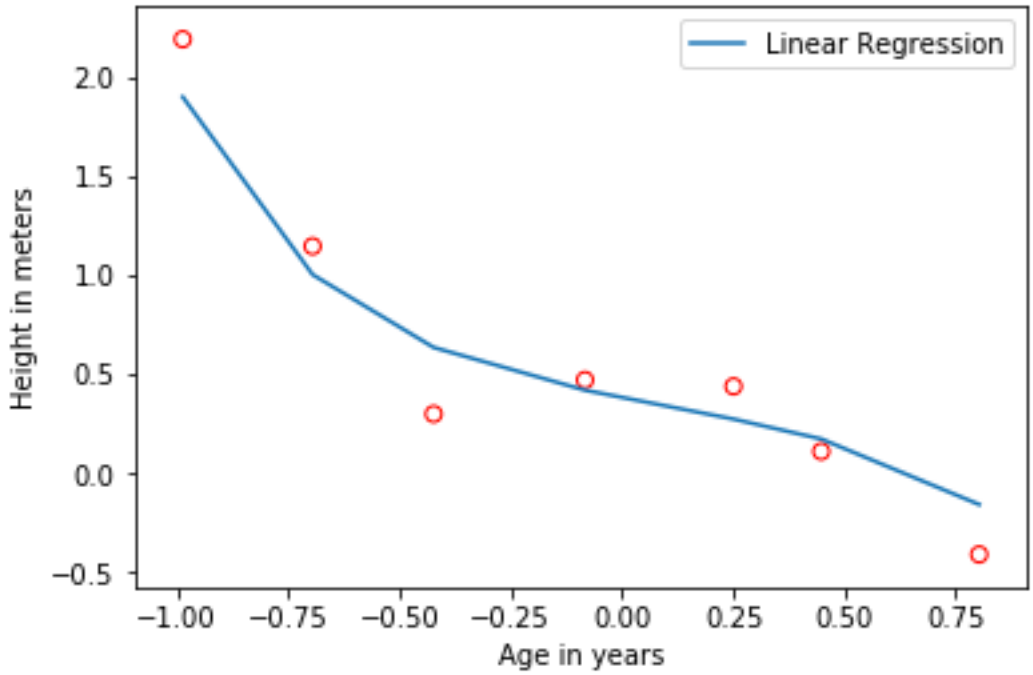
theta: array([0.49917479, 0.83306615, -2.03845857, -6.93424088, 3.39424083, 5.75306609])

L2Norm: 9.8894299563366523

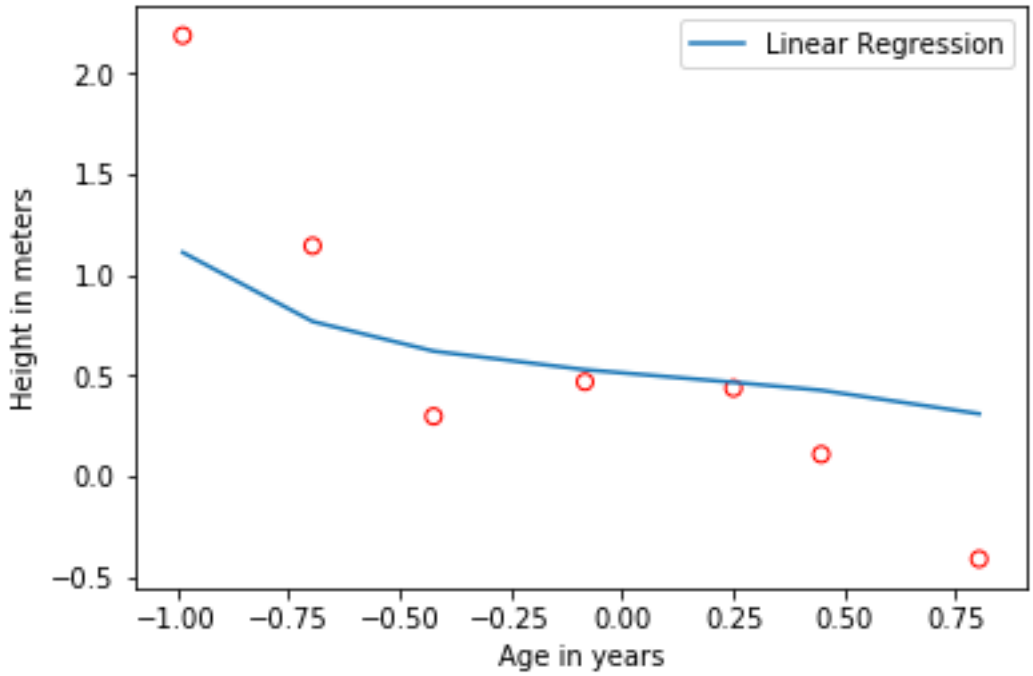


lambda: 1

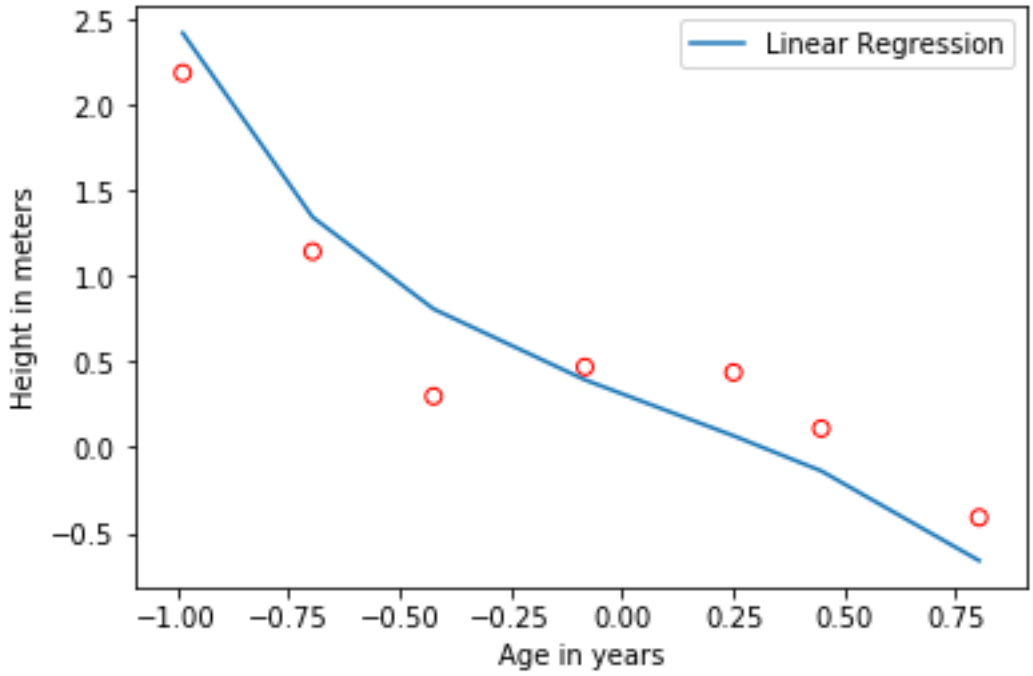
theta: array([0.38106374, -0.43781896, 0.12880159, -0.43609185, 0.1933533 , -0.37688109])
L2Norm: 0.85034452774484648



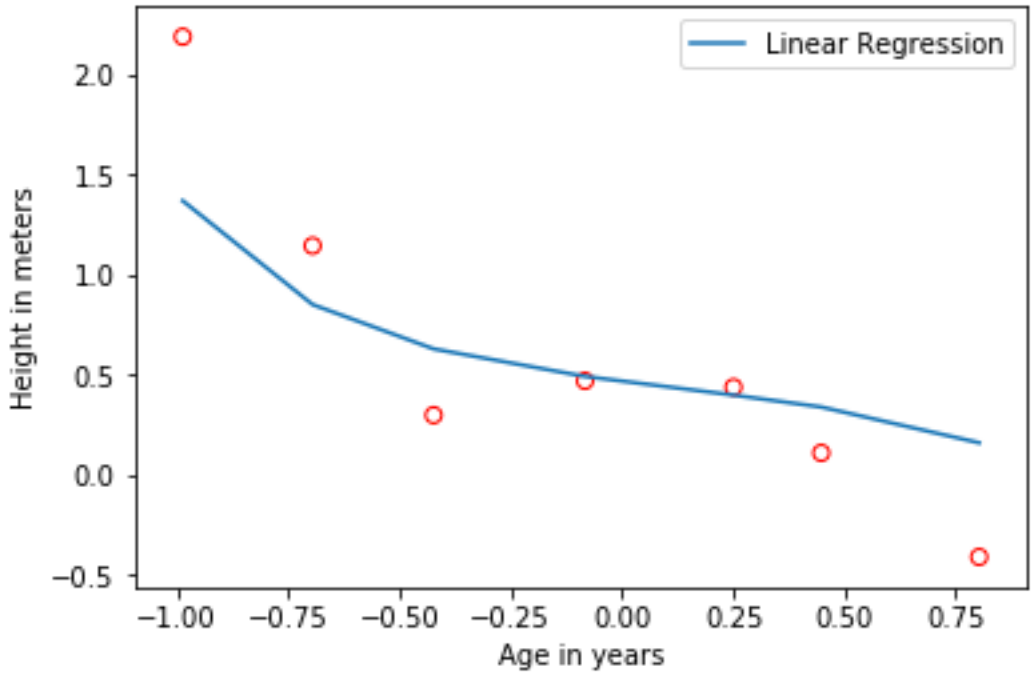
lambda: 10
theta: array([0.51357264, -0.19100471, 0.06437106, -0.15400044, 0.07903144, -0.13137918])
L2Norm: 0.59296363841252353



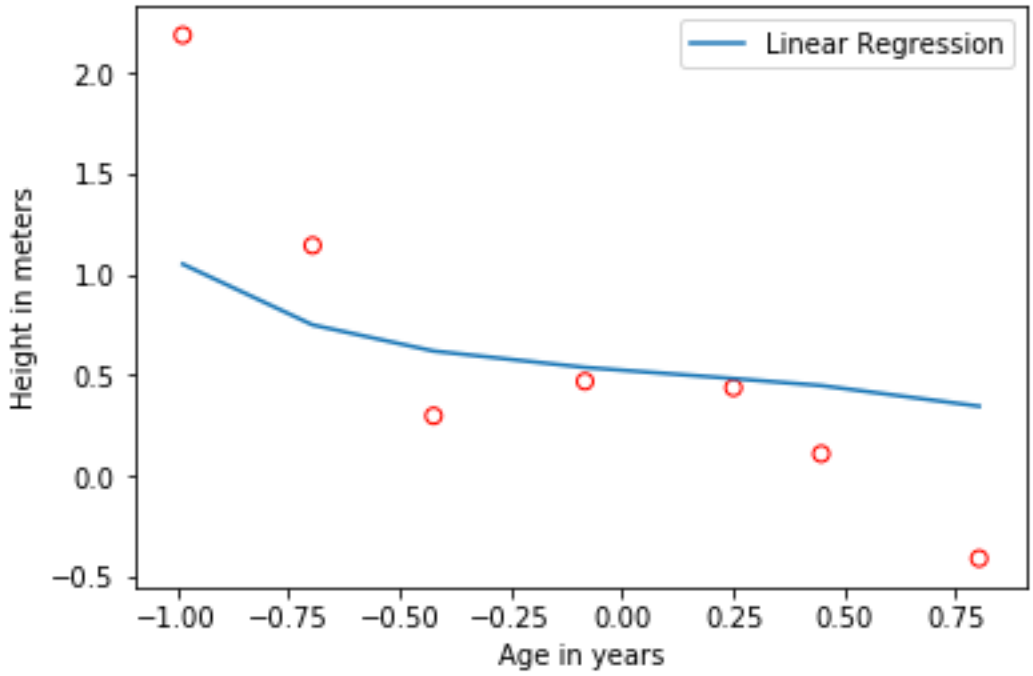
lambda: -1
theta: array([0.31089401, -0.98611001, 0.18116587, -0.48964457, 0.16984902, -0.34777078])
L2Norm: 1.2212428600852612



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.

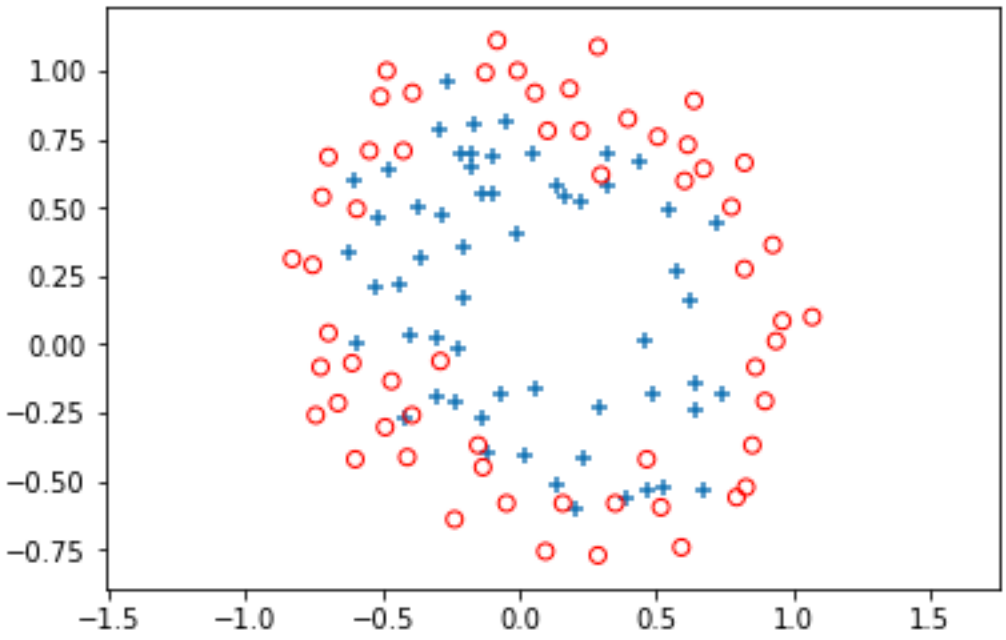
Code:

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 18 20:32:55 2017

@author: rditljtd
"""

import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from map_features import *

x = np.loadtxt('ax.dat')
y = np.loadtxt('ay.dat')

#Plot original data out
plt.scatter(x, y, facecolors='none', color='red')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()

#Get the number of examples
m = x.shape[0]
#Reshape x to be a 2D column vector
x.shape = (m,1)
#Add a column of ones to x
X = np.hstack([np.ones((m,1)), x, x**2, x**3, x**4, x**5])

theta = np.zeros(shape=(6,1))

lambdaa = [0, 1, 10, -1, 5, 12]

#L2norm = np.linalg.norm(X)

#print L2norm

for i in range(len(lambdaa)):

    matrix = np.diag([0, 1,1,1,1,1])
    #print matrix

    theta = np.linalg.inv(X.transpose().dot(X) + lambdaa[i]*(matrix)).dot(X.tran
spose().dot(y))
    print "lambda: " + `lambdaa[i]`
    print "theta: " + `theta`
    print "L2Norm: " + `np.linalg.norm(theta)`
    #Plot original data out
    plt.plot(X[:,1],np.dot(X,theta))
    plt.legend(['Linear Regression', 'Training Data'])
    plt.scatter(x, y, facecolors='none', color='red')
    plt.xlabel('Age in years')
    plt.ylabel('Height in meters')
    plt.show()
    i = i+1

#def computeCost(X, y, theta):
```

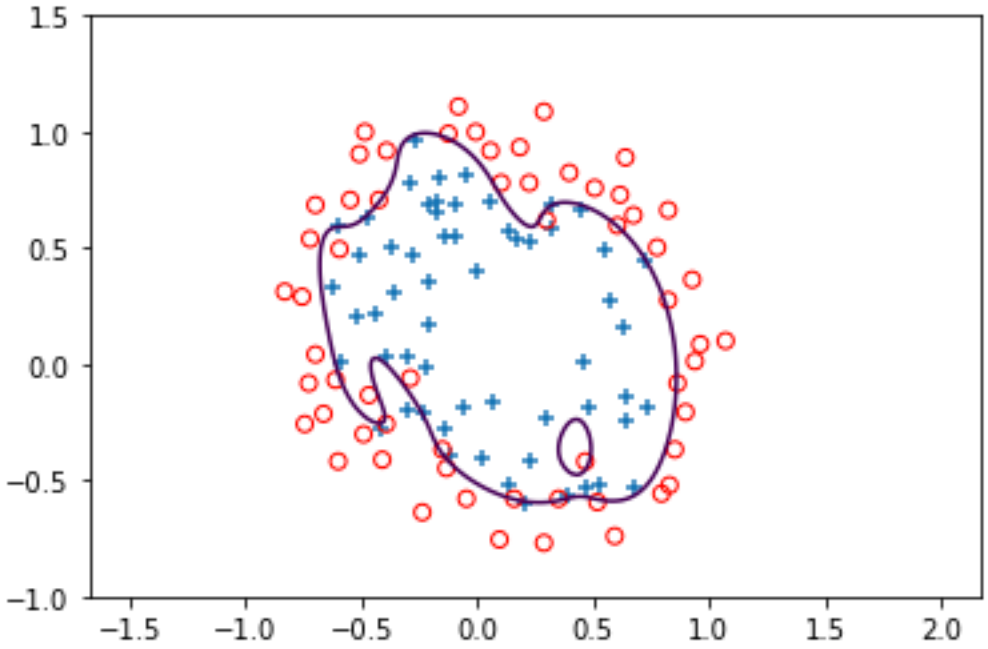
```
#    m=y.size
#    estimates = (X-y[:,None]).dot(theta).flatten()
#    squared_errors = (estimates) ** 2
#    sum_errors = (1 / (2 * m))*squared_errors.sum()
#    J = lambdaa*(theta)
#    return J
#
#    #sum_first = ((hypothesis(X)-y)**2).sum()
#    #sum_second = lambdaa*((theta**2).sum)
#    #J_theta = (1/(2*m))*(sum_first+sum_second)
#
#def gradientDescent(X, y, theta, lambdaa, iterations):
#    m = y.size
#    J_history = np.zeros(shape = (iterations, 1))
#
#    for i in range(iterations):
#        J_history[i,0] = computeCost(X, y, theta)
#
#    return theta, J_history
#
##theta, J_history = gradientDescent(X, y, theta, lambdaa, 10)
#
#print theta
```

Part B:

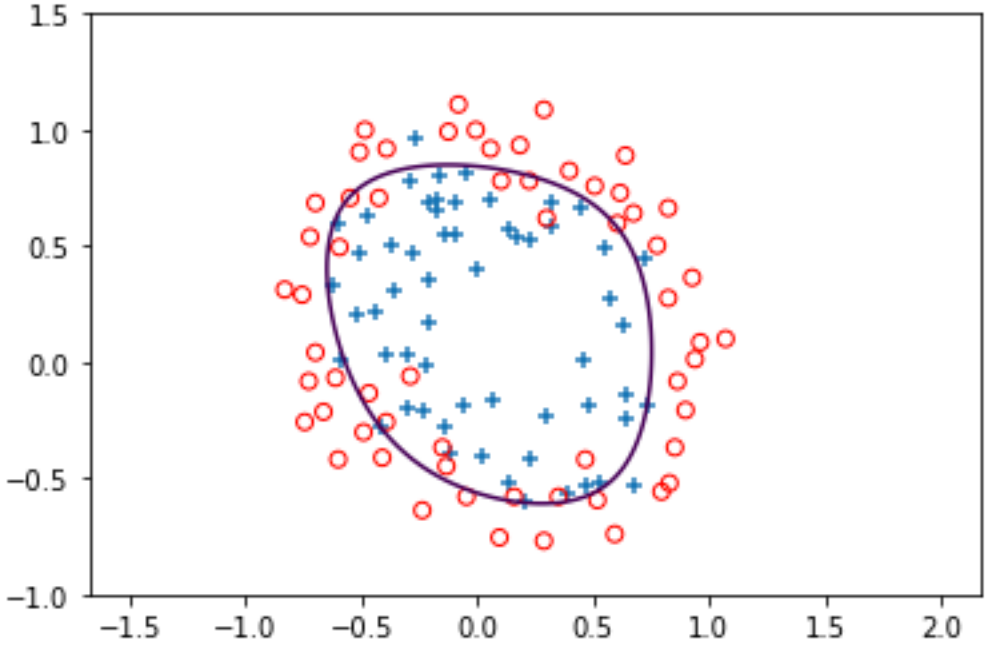
Plot of Raw Data:



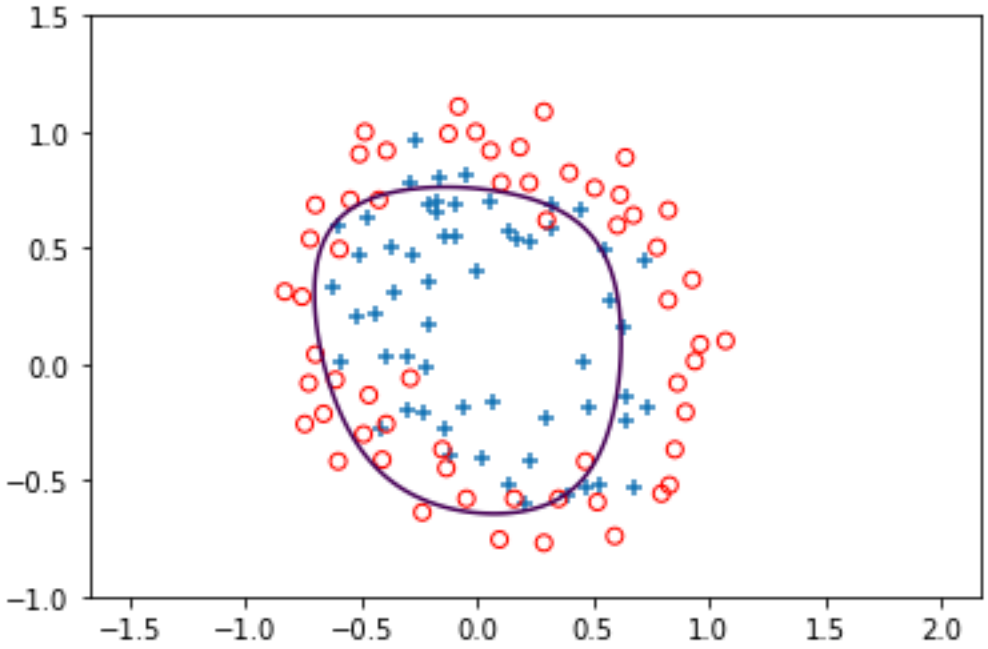
Final theta: [[27.04247528] [19.82536158] [68.94141094] ..., [-2670.0856728] [-1307.90297298] [-522.34199678]]
Lambda: 0
L2Norm: 7301.09758994



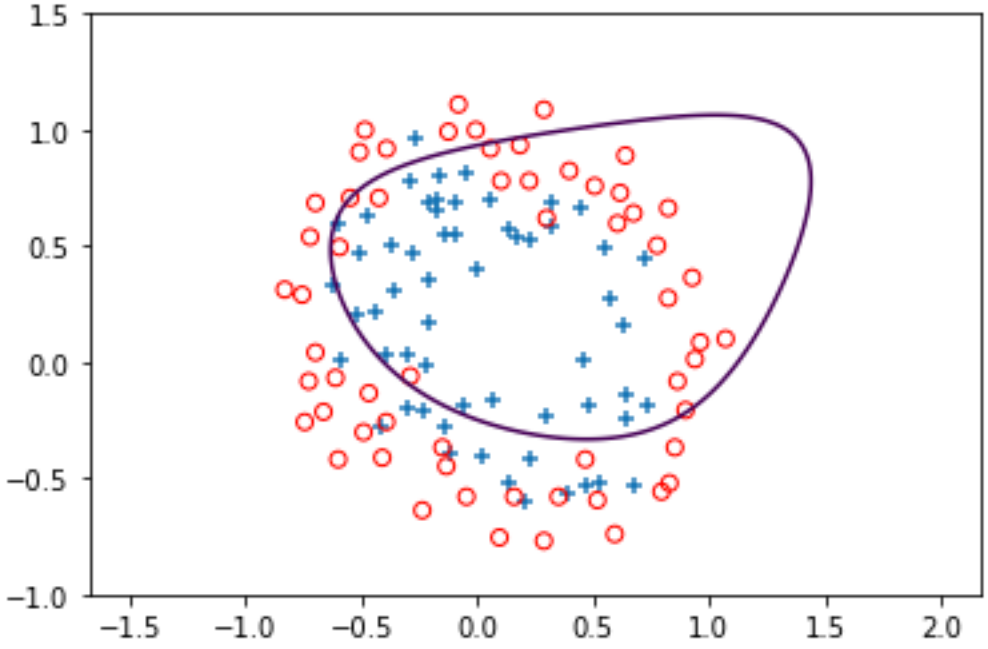
Final theta: [[1.18455196] [0.68338187] [1.17726894] ..., [-0.33930754] [-0.13867704] [-0.92708495]]
Lambda: 1
L2Norm: 4.03969126766



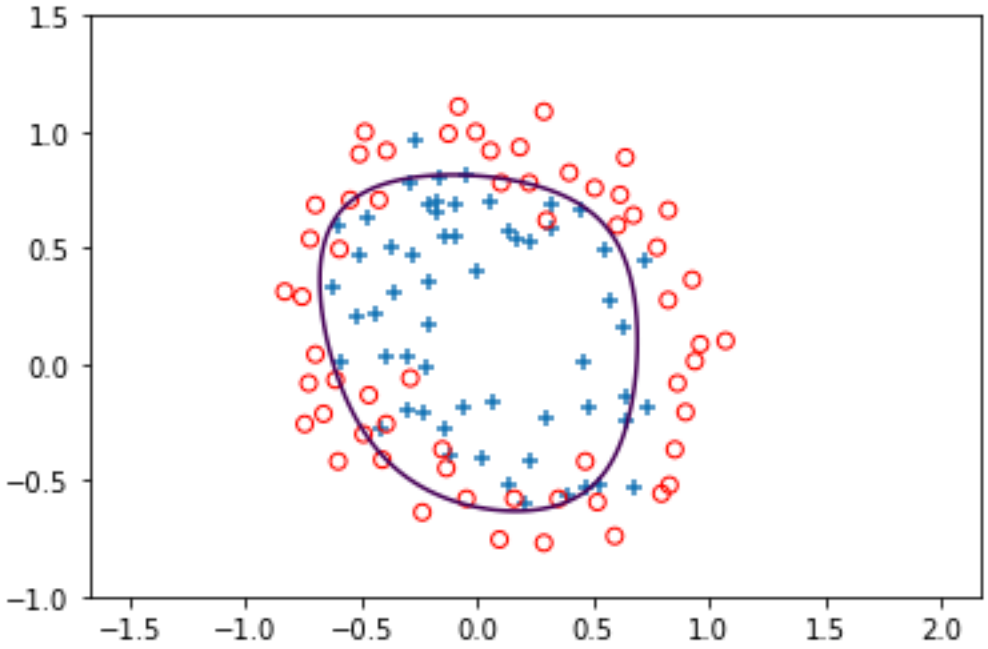
Final theta: [[0.22865286] [0.00972808] [0.17478993] ..., [-0.06220219] [-0.01216701] [-0.2639623]]
Lambda: 10
L2Norm: 0.84144167998



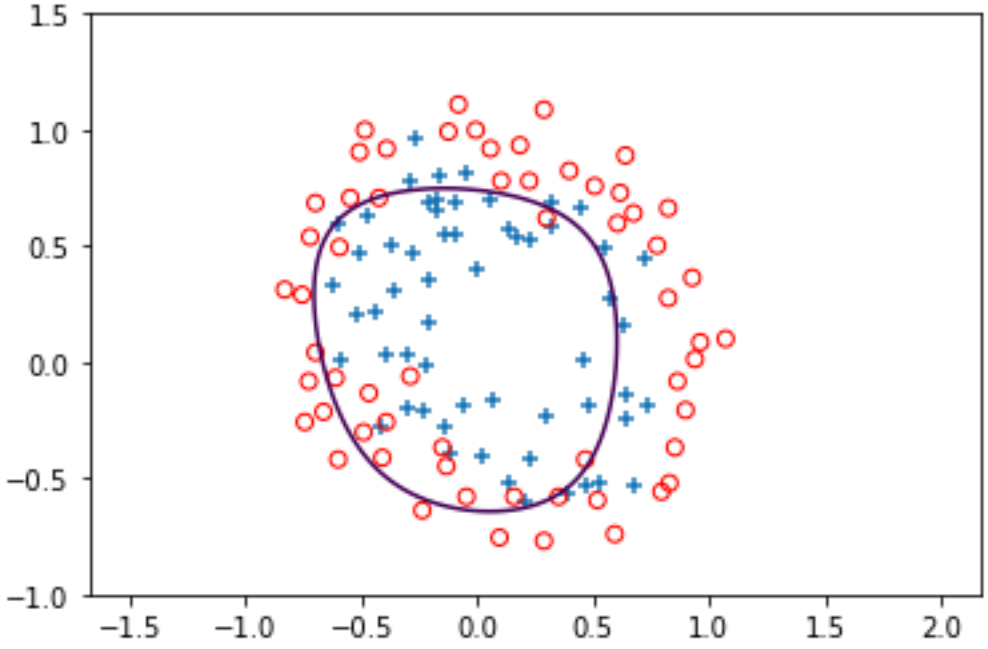
Final theta: [[-1.79501504] [-3.1821499] [-6.62131099] ..., [0.45324879] [-0.57963481] [3.5393426]]
Lambda: -2
L2Norm: 10.2832842969



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:

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 21 20:23:00 2017

@author: rditljtd
"""

import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from scipy.special import expit
from map_features import *

x = np.loadtxt('bx.dat', delimiter=",")
y = np.loadtxt('by.dat')


##Get the number of examples
#m = x.shape[0]
#
##Reshape x to be a 2D column vector
#x.shape = (m,2)
#
##Add a column of ones to x
#X = np.hstack([np.ones((m,1)), x])
#
##initialize theta
theta = np.zeros(shape=(28,1)) #Initialize theta
##print theta
##alpha = .3#Your learning rate#
##J = []
#iterations = 15
#J_history = np.zeros(shape = (iterations, 1))

#pos = np.nonzero(y)
#neg = np.where(y==0)[0]
#
#plt.scatter(x[pos,0],x[pos,1],marker='+')
#plt.scatter(x[neg,0],x[neg,1],facecolors='none',marker='o', color='r')
#plt.show()

#Find indices of positive and negative examples
pos = np.nonzero(y)
neg = np.where(y==0)[0]
#Plot out the data
plt.scatter(x[pos,0],x[pos,1],marker='+')
plt.scatter(x[neg,0],x[neg,1],facecolors='none',marker='o',color='r')
```

```
plt.axis('equal')
plt.show()

X=map_features(x[:,0], x[:,1])
#Newtons method solution

def sigmoid(z):
    # print "z: " + `z`
    toreturn = expit(z)
    # print "sigmoid: " + `toreturn`
    return toreturn

def hypothesis(X, theta):
    # print "here: " + `theta`
    # print "there: " + `X`
    toreturn = (sigmoid(np.dot(X, theta)))
    # print "hypothesis: " + `toreturn`
    return toreturn

def computeCost(X, y, theta, lambdaa):
    m = y.size
    firstOp = ((1.0/m) * (-y*np.log(hypothesis(X, theta))) - (1.0-
y)*(np.log(1.0-hypothesis(X, theta)))).sum()
    toreturn = ((1.0*lambdaa/(2.0*m)) * (theta**2.0)).sum() - firstOp
    #print "computeCost: " + `toreturn`
    return toreturn

def gradientDescent(X, y, theta, lambdaa):
    m = y.size
    minus = hypothesis(X, theta) - y
    result = (1.0/m)*(X.T.dot(minus))
    toreturn = ((lambdaa*1.0)/m) * theta + (result)
    #print "gradientDescent: " + `toreturn`
    #print "result: " + `result`
    return toreturn

def hessian(X, y, theta, lambdaa):
    m = y.size
    minus = (1.0-hypothesis(X,theta))*(hypothesis(X, theta))
    Xm = minus*X
    result = (1.0/m)*(Xm.T.dot(X))
    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])

    toreturn = ((lambdaa*1.0)/m) * matrix + (result)
    # print "hessian: " + `toreturn`
    #result and toreturn are the same?
    return toreturn

def newtonsMethod(X, y, theta, lambdaa, iterations):
    m=y.size
    y.shape = (m,1)
    for i in range(iterations):
        J_history[i] = computeCost(X, y, theta, lambdaa)
                theta = (theta -
    np.linalg.inv((hessian(X, y, theta, lambdaa))).dot(gradientDescent(X, y, theta,
    lambdaa)))
            #print theta, J_history
            return theta, J_history

       def runNewtonsMethod(l):
            lambdaa = l
```

```
            theta1, J_history = newtonsMethod(X, y, theta, lambdaa, 15)
            print 'Final theta: ', theta1
            print 'Lambda: ' + `lambdaa` + ' L2Norm: ', np.linalg.norm(theta1)

            #Define the ranges of the grid
            u = np.linspace(-1,1.5,200)
            v = np.linspace(-1,1.5,200)
            #Reshape to be 2-D
            u.shape = (len(u),1)
            v.shape = (len(v),1)
            #Plotting commands below
            X1, Y = np.meshgrid(u,v)
            Z = np.zeros((len(u),len(v)))
            #Initialize z = theta*x over the whole grid
            for i in range(len(u)):
                for j in range(len(v)):
                    Z[j][i] = np.dot(map_features(u[i],v[j]),theta1)
            plt.clf()
            plt.scatter(x[pos,0],x[pos,1],marker='+')
            plt.scatter(x[neg,0],x[neg,1],facecolors='none',marker='o',color='r'
    )
            plt.axis('equal')
            plt.contour(X1,Y,Z, 0,linewidth=2)
            plt.show()

        i=0
        lambdaas=[0, 1, 10, -2, 4, 12]
        for i in range(len(lambdaas)):
            runNewtonsMethod(lambdaas[i])
```