Machine Learning
CSE 8673

Programming Assignment 2

Student: Anh Do
NetID: aqd14

Part A: Deliverables

Please include in your project write up the following plots and answers to questions

1. Plot of raw data

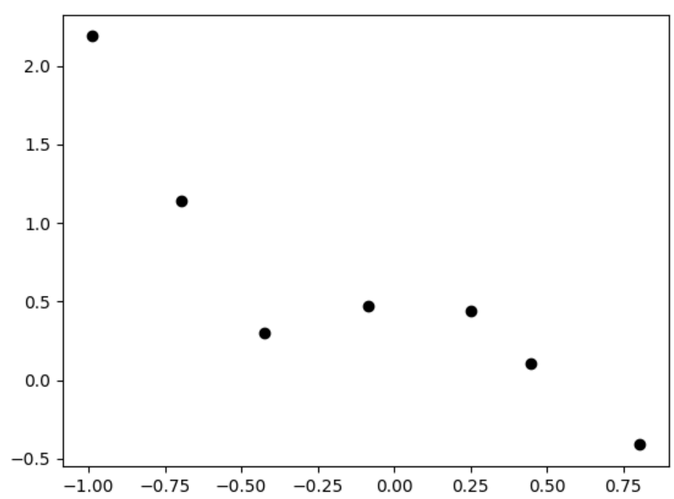


Figure 1. Plot raw data for part A

2. For each different λ value (3 required, 3 you choose), report:
(a) What value of θ did you get?

Lambda	**Theta**
0	[0.49917479 0.83306615 -2.03845857 -6.93424088 3.39424083 5.75306609]
1	[0.38106374 -0.43781896 0.12880159 -0.43609185 0.1933533 -0.37688109]

5	[0.46756365 -0.28432857 0.09426909 -0.23548116 0.1184942 -0.20138849]
10	[0.51357264 -0.19100471 0.06437106 -0.15400044 0.07903144 -0.13137918]
100	[0.59266179 -0.02738086 0.00936885 -0.02153072 0.01125834 -0.01832202]
1,000,000	[6.05801478e-01 -2.87490404e-06 9.85392959e-07 -2.25426435e-06 1.18128378e-06 -1.91774687e-06]

(b) What is the L2-norm of the θ value you got?

Lambda	**L2-norm**
0	9.889430
1	0.850345
5	0.646834
10	0.592964
100	0.594148
1,000,000	0.605801

(c) A plot of the learned function. These could all be on the plot.

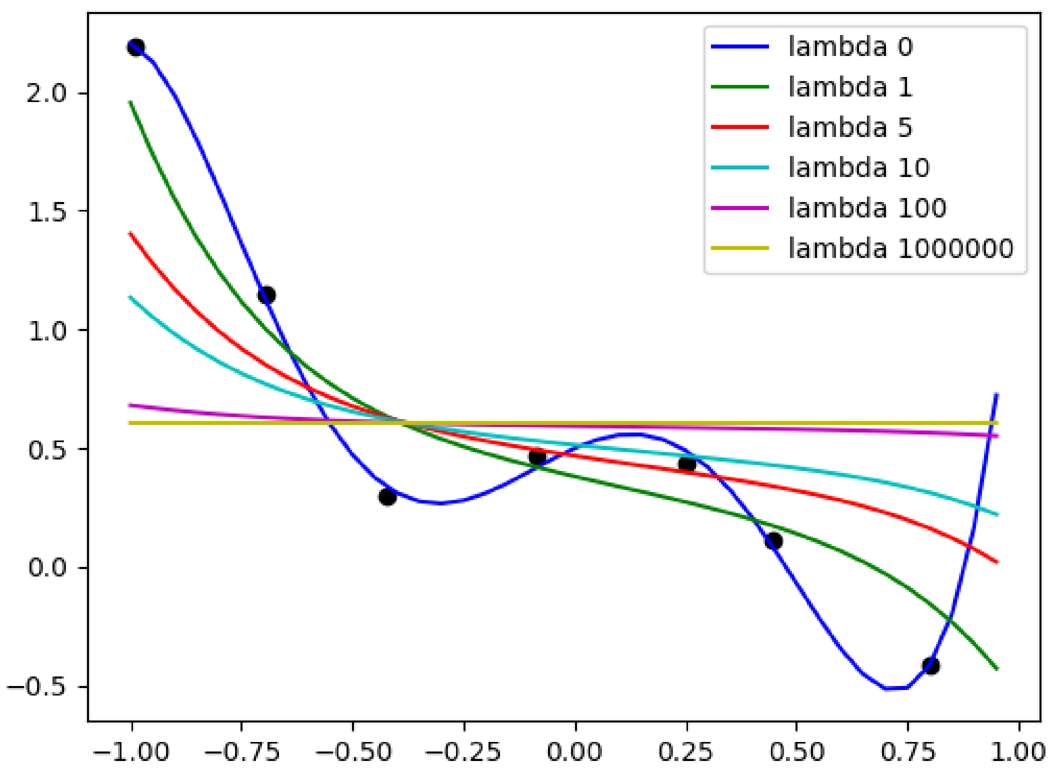


Figure 2. Plot of learned model with different lambda

(d) Discuss how well you expect this θ to generalize to other data points drawn from the same distribution. (We don't have other data, so this will be based on what you think the entire distribution will look like)

Answer: The θ with λ = 0 will probably suffer from overfitting problem. The machine learning model is somewhat complex and only works by memorizing the seen training data. See **3c** for more details.

3. Answer the following questions:

(a) How the regularization parameter λ affects your model?

Answer: Regularization parameter λ adjust the complexity of the machine learning model by penalizing our weights vector (i.e., by using L2-norm).

(b) What would our model look like as $\lambda \rightarrow \infty$?

Answer: When $\lambda \rightarrow \infty$, our thetas vector values become extremely small. Therefore, our model becomes a linear function of bias.

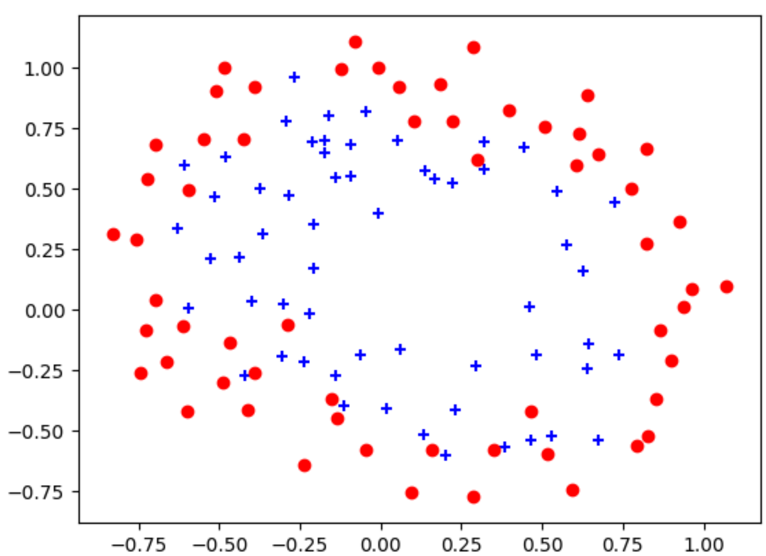
(c) What λ do you think results in the model that will generalize the best?

Answer: For chosen lambda value, I think the model with λ = 1 will generalize the best. The model with λ = 0 faces the overfitting problem because it only memorizes the training data. The model with λ = 1 almost matches with some training data but also be curving enough to predict the previously unseen data. Other lambda values suffer from underfitting.

Part B: Deliverables

Please include in your project write up the following plots and answers to questions

1. Plot of raw data



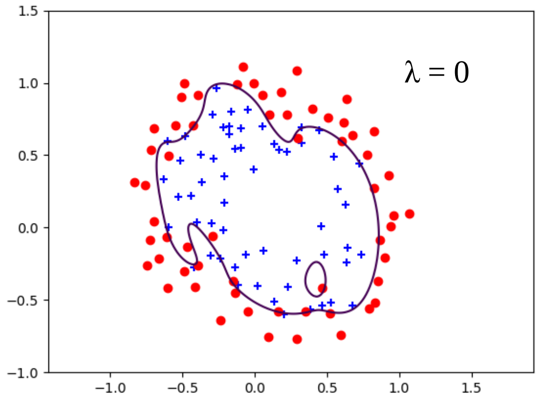
2. For each λ value (3 required, 3 you choose), please report:

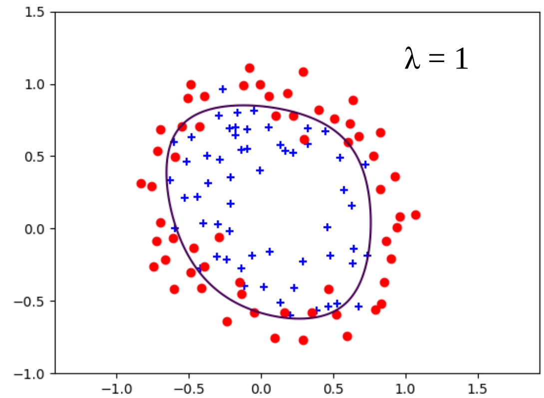
- What was the L2-norm of the θ values you got?
- What values of θ did you get?
- How many iterations were required for convergence?
- Plot of final decision boundary found
- What does this model output for an input of (0.5, 0.5)?

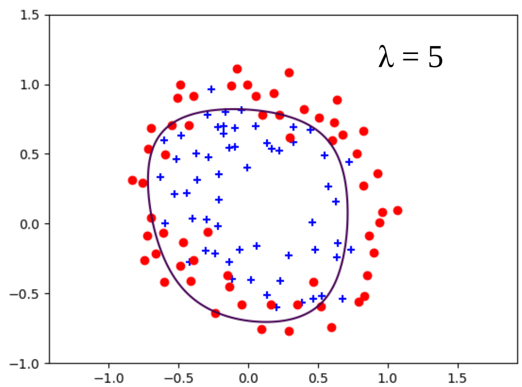
Answer:

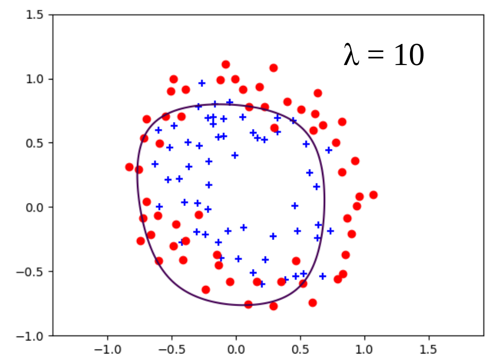
Lambda	L2-norm	Iterations	Theta	Output
0	7301.097590	16	27.04247528 19.82536158 68.94141094 -323.10507475 -158.16441326 -108.60656083 -132.96913484 -492.61726783 -406.58660701 -363.03142451 1318.61571797 1384.19893953 1381.09959651 584.07341975 276.6374887 351.98491848 1118.55333839 1416.69770844 1675.23155955 1128.91948372 606.88766459 -1506.60749207 -2343.61431421 -3424.25359631 -3192.12272451 -2670.0856728 -1307.90297298 -522.34199678	1
1	4.245714	5	1.32241821 0.71137181 1.19220868 -1.99133437 -0.92146984 -1.51861566 0.12805279 -0.37247721 - 0.41142477 -0.1658268 -1.47651984 -0.0528076 -0.64895942 -0.27608993 - 1.20705105 -0.27541198 -0.20896745 -0.06394606 -0.28104152 -0.3124246 - 0.44554571 -1.07948128 0.0260809 -0.30720874 0.01552269 -0.33790573 - 0.14510699 -0.91950837	0.677492
5	1.565187	4	5.52959564e-01 1.12060394e-01 3.50893311e-01 -7.58372612e-01 -2.15950575e-01 -4.93599665e-01 -5.60269648e-02 -1.06696603e-01 -1.19935434e-01 -1.35328958e-01 -5.67910332e-01 -2.22606478e-02 -2.12359874e-01 -5.49706765e-02 -4.70921406e-01 -1.61132815e-01 -6.73816069e-02 -3.66259610e-02 -8.71742596e-02 -7.93511031e-02 -2.70210496e-01 -4.22396095e-01 -2.53043970e-03 -1.04442736e-01 1.04778842e-04 -1.13010359e-01 -2.40404359e-02 -4.25906303e-01	0.560497
10	0.940065	4	3.48390005e-01 9.87871398e-03 1.63730352e-01 -4.43951809e-01	0.528019

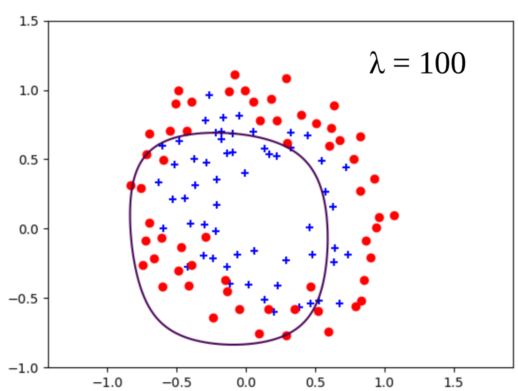
			-1.10600451e-01 -2.89727235e-01 -6.75598999e-02 -5.98131907e-02 -6.76169068e-02 -1.07578810e-01 -3.38441852e-01 -1.32292014e-02 -1.20233033e-01 -2.69183994e-02 -2.90035454e-01 -1.17875949e-01 -3.80689126e-02 -2.35923965e-02 -4.97740854e-02 -4.21632552e-02 -1.87485489e-01 -2.56961251e-01 -3.38417324e-03 -5.94689634e-02 -4.00667308e-04 -6.50143643e-02 -1.12331303e-02 -2.73187585e-01	
100	0.126803	3	0.03946607 -0.01475491 0.00486043 -0.05472123 -0.01289358 -0.0399837 -0.01764224 -0.00804877 -0.00892722 -0.02336593 -0.04344499 -0.00235626 -0.01452437 -0.00326997 -0.04217347 -0.0209109 -0.00486706 -0.00363217 -0.00647767 -0.00494631 -0.03242649 -0.03438026 -0.0011188 -0.00716899 -0.00035226 -0.0081638 -0.00137826 -0.04159813	0.497500
1,000,000	-0.017094	2	-1.70878689e-02 -1.86579240e-06 6.80307556e-08 -5.64175603e-06 -1.35406925e-06 -4.29198097e-06 -2.01189155e-06 -8.68650822e-07 -9.45448884e-07 -2.69841464e-06 -4.51508266e-06 -2.68833002e-07 -1.50437457e-06 -3.51400850e-07 -4.55141344e-06 -2.28682285e-06 -5.14986936e-07 -3.92341452e-07 -6.88052793e-07 -5.13016227e-07 -3.60055197e-06 -3.59942632e-06 -1.34220796e-07 -7.40374674e-07 -4.61658323e-08 -8.53798260e-07 -1.49696628e-07 -4.51420875e-06	0.495727

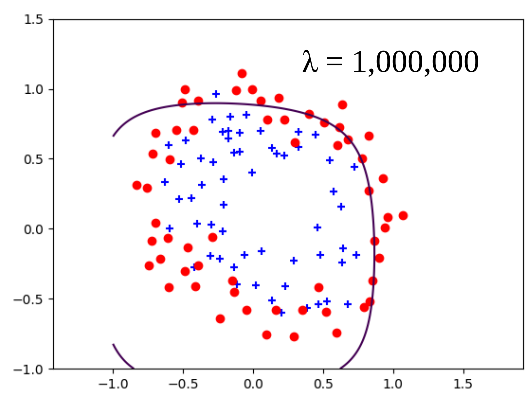

λ = 0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5


λ = 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5


λ = 5
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5


λ = 10
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5


λ = 100
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5


λ = 1,000,000
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5

3. What happens to the decision boundary as λ is increased?
Answer: when λ is increased, it becomes more difficult to classify input data. When it gets to 1,000,000 the decision boundary even can't be completed

4. What value of λ appears to give the best decision boundary (in terms of generalization)?
Answer: It looks like with λ = 1 we have the best decision boundary. It contains least wrong classification (except the model with λ = 0, which is overfitting).

5. Be sure to include in your .pdf submission your code that you wrote for this assignment.

Appendix

Part A

regulalized_linear_regression.py

```
'''
Created on Sep 17, 2017

@author: doquocanh-macbook
'''

import numpy as np
from numpy.linalg import inv, norm
# from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

X = np.loadtxt('pa2data/ax.dat')
y = np.loadtxt('pa2data/ay.dat')

# print(X)
# print(y)
plt.scatter(X, y, facecolors='black')
# plt.show()

# number of training data
m = X.shape[0]

# increase model capacity by adding higher polynomial
X = np.stack((np.ones(m), X, X**2, X**3, X**4, X**5), axis=-1)

# number of feature
n = X.shape[1]
diagonal_matrix = np.diag(np.ones(n))
```

```
diagonal_matrix[0][0] = 0

_lambdas = np.array([0, 1, 5, 10, 100, 1000000])
thetas = []

for i in range(_lambdas.size):
    theta = inv(X.T.dot(X) + _lambdas[i] * diagonal_matrix)
    theta = theta.dot(X.T)
    theta = theta.dot(y)
    thetas.append(theta)

# calculate norm for thetas
for i in range(len(thetas)):
    print('Lambda = %d \t L2-norm = %f' % (_lambdas[i], norm(thetas[i])))

# input value range
r = np.arange(-1,1,0.05)

features = np.stack((np.ones(r.shape[0]), r, r**2, r**3, r**4, r**5), axis=-
1)
# plot data
colors = ['b', 'g', 'r', 'c', 'm', 'y']
for i in range(_lambdas.size):
    print(thetas[i])
    plt.plot(r, features.dot(thetas[i]), color=colors[i], label='lambda ' +
str(_lambdas[i]))

plt.legend(loc='upper right')
plt.show()
```

Part B

regulazied_logistic_regression.py

```
'''
Created on Sep 19, 2017

@author: aqd14
'''

import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import inv, norm
from map_features import *
```

```python
def newton_method(X, y, _lambda, tolerance=1e-5, max_iters=20):
    theta = np.zeros((X.shape[1], 1))
    epoch = 1
    for _ in range(max_iters):
        H = regularized_hessian(X, theta, _lambda)
        # print(hes)
        g = regularized_gradient(X, y, theta, _lambda)
        # print('Gradient shape: ', g.shape)
        temp = theta - np.dot(inv(H), g)
        if np.sum(abs(theta - temp)) < tolerance:
            print('Convergered at epoch %d' % epoch)
            break
        theta = temp
        epoch += 1
    if epoch >= max_iters:
        print('Reached maximum iteration!')
    return theta

def regularized_gradient(X, y, theta, _lambda):
    m = X.shape[0]
    h = hypothesis(X, theta)
    g = (1.0/m) * X.T.dot(h-y)
    # Adjust result with regularization parameter
    g[1:] += (_lambda * theta[1:])/m
    return g

def regularized_hessian(X, theta, _lambda):
    m = X.shape[0]
    h = hypothesis(X, theta)
    h.shape = (len(h),)
    H = (1.0/m) * np.dot(np.dot(X.T, np.diag(h)), np.dot(np.diag(1-h), X))
    # Adjust result with regularization parameter
    reg_diag_matrix = np.diag(np.ones(X.shape[1]))
    reg_diag_matrix[0][0] = 0
    H += (_lambda * reg_diag_matrix)/m
    return H

def sigmoid(z):
    result = 1.0/(1.0+np.exp(-z))
    return result

def regularized_cost_function(X, y, theta, _lambda):
    """Calculate cost function with sigmoid activation function and 
regularization
```

```
    Parameters
    ----------
    X : array-like
        Training input data
    y : array-like
        Training output data
    """
    m = X.shape[0]
    h = hypothesis(X, theta)
    J = (_lambda * theta[1:]**2)/(2*m) + (1.0/m) * (-y.dot(np.log(h)) - (1-
y).dot(np.log(1-h)))
    return J

def hypothesis(X, theta):
    # print('Hypothesis: ', h.shape)
    return sigmoid(X.dot(theta))

def main():
    # Load dataset
    X = np.loadtxt('pa2data/bx.dat', delimiter=',')
    y = np.loadtxt('pa2data/by.dat')

    # Find indices of positive and negative examples
    pos = np.nonzero(y)[0]
    neg = np.where(y==0)[0]

    # Plot out the raw data
    '''
    plt.scatter(X[pos, 0], X[pos, 1], marker="+", color="b")
    plt.scatter(X[neg, 0], X[neg, 1], marker="o", color="r")
    plt.show()
    '''

    # 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
    X_axis, Y_axis = np.meshgrid(u, v)
    Z = np.zeros((len(u), len(v)))
```

```python
    # Prepare data for Newton method
    # Create more features for our training data with feature mappings
    X_added_features = map_features(X[:, 0],X[:,1])
    # m = X.shape[0]
    # X = np.column_stack((np.ones((m, 1)), X))
    y.shape = (y.shape[0], 1)
    _lambdas = np.array([0, 1, 5, 10, 100, 1000000])

    test_data = map_features(np.array([0.5]), np.array([0.5]))
    for t in range(_lambdas.size):
        theta = newton_method(X_added_features, y, _lambdas[t])
        print('Theta value = %s\n' % theta[:,0])
        print('Lambda = %d \t L2-norm = %f\n' % (_lambdas[t], norm(theta)))
        print('Prediction value for input (0.5, 0.5) is %f' %
hypothesis(test_data, theta))
        print('------------- END ---------------')
        for i in range(len(u)):
            for j in range(len(v)):
                Z[i][j] = np.dot(map_features(u[i], v[j]), theta)

        plt.clf()
        plt.scatter(X[pos,0], X[pos,1], marker='+', color='b')
        plt.scatter(X[neg,0], X[neg,1], marker='o', color='r')
        plt.axis('equal')
        plt.contour(X_axis, Y_axis, Z.T, 0, linewidth=2)
        plt.show()

if __name__ == '__main__':
    main()
```