CS 8673 Programming Assignment 2: Regularization

September 15, 2017

This programming assignment is due on Wednesday, September 27, 2017, at 11:59pm.

Honor code note: You are welcome to discuss the assignment with other students, but you must individually and independently write the code and other deliverables that you submit for the assignment. Any copying of code from outside sources or other students will be considered an honor code violation.

---

1 Introduction

The folder you are provided contains:

- `PA2_Handout.pdf` - This file that you are reading
- `pa2data` - This is a folder containing the data files you will use for this assignment.

This assignment will continue your exposure to some simple machine learning algorithms. This assignment uses the same programming environment (numpy, scipy, matplotlib) as the previous one. Again, if you prefer to work in matlab or octave you are welcome to do so.

1.1 Credit

These assignments are based off of assignments from Andrew Ng's Stanford class, which you can find at:

http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=MachineLearning

Instructions there are for matlab/octave, so please reference them if you want to work in matlab. Solutions are available online for matlab, but please make sure that any code you submit is your own. The data for this assignment has been modified from the data used in that course, so the answers are different.

1.2 Deliverables

Please turn in your submission as a *single* `.pdf` *file* with answers to all of the relevant questions as well as the plots that you need and a code appendix with your solutions. This will make the grading process much easier.

1.3 Data

Download `pa2data.zip`, and extract the files from the zip file. They should end up in a folder called `pa2data`. This data bundle contains two sets of data, one for linear regression and the other for logistic regression. It also includes a helper function named `map_features.py` which will be used for logistic regression. Make sure that this file is placed in the same working directory where you plan to write your code.

1.4 Code Imports

All of the code examples assume that the following import statements have been executed (put at the top of your python file).

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

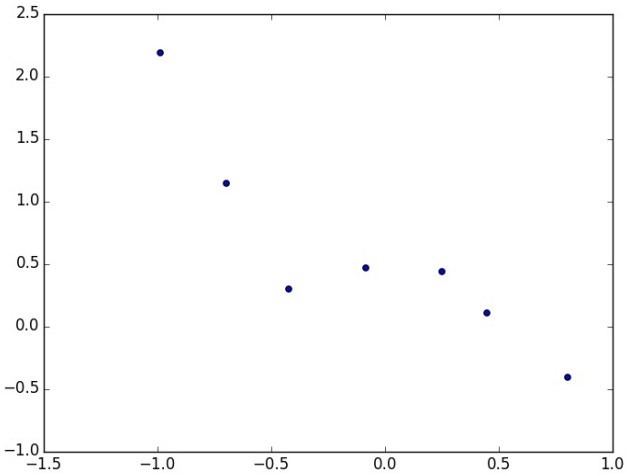
2 Part A: Regularized Linear Regression

Part A focuses on regularized linear regression and the normal equations.

2.1 Plot the data

Load the data files `ax.dat` and `ay.dat` into your program. These correspond to the `x` and `y` variables that you will start out with.

Notice that in this data, the input `x` is a single feature, so you can plot `y` as a function of `x` on a 2-dimensional graph.



From looking at this plot, it seems that fitting a straight line might be too simple of an approximation. Instead, we will try fitting a higher-order polynomial to the data to capture more of the variations in the points.

2.2 Regularized Linear Regression

Let's try a fifth-order polynomial. Our hypothesis will be:

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \theta_5 x^5$$

This means that we have a hypothesis of six features, because $x^0, x^1, \ldots x^5$ are now all features of our regression. Notice that even though we are producing a polynomial fit, we still have a linear regression problem because the hypothesis is linear in each feature. Since we are fitting a 5th-order polynomial to a data set of only 7 points, over-fitting is likely to occur. To guard against this, we will use regularization in our model.

Recall that in regularization problems, the goal is to minimize the following cost function with respect to θ:

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

The regularization parameter λ is a control on your fitting parameters. As the magnitudes of the fitting parameters increase, there will be an increasing penalty on the cost function. This penalty is dependent on the squares of the parameters as well as the magnitude of λ. Also, notice that the summation after λ does not include θ_0^2.

2.3 Normal equations

Now we will find the best parameters of our model using the normal equations. Recall that the normal equations solution to regularized linear regression is

$$\theta = (X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix})^{-1} X^T \vec{y}$$

The matrix following λ is an $(n+1) \times (n+1)$ diagonal matrix with a zero in the upper left and ones down the other diagonal entries. (Remember that n is the number of features, not counting the intercept term). The vector $\vec{y}$ and the matrix X have the same definition they had for unregularized regression:

$$\vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix} \quad X = \begin{bmatrix} —(x^{(i)})^\top— \\ —(x^{(i)})^\top— \\ \vdots \\ —(x^{(i)})^\top— \end{bmatrix}$$

Using this equation, find values for θ using the three regularization parameters below:

1. $\lambda = 0$ (this is the same case as non-regularized linear regression)
2. $\lambda = 1$
3. $\lambda = 10$

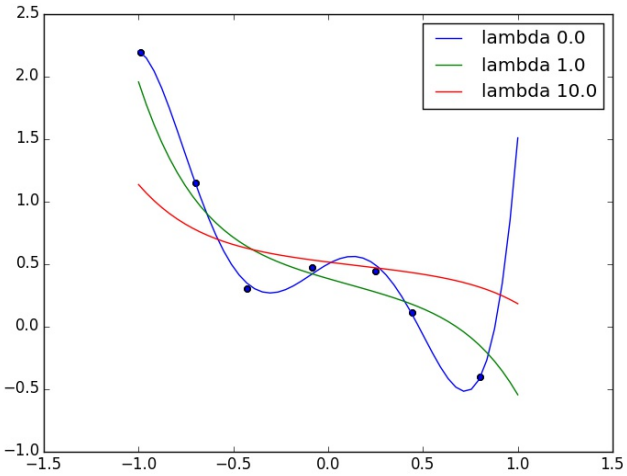
As you are implementing your program, keep in mind that X is an $m \times (n+1)$ matrix, because there are m training examples and n features, plus an $x_0 = 1$ intercept term. In the data provided for this exercise, you were only give the first power of x. You will need to include the other powers of x in your feature vector X, which means that the first column will contain all ones, the next column will contain the first powers, the next column will contain the second powers, and so on. You can do this in numpy with the command

```
x = np.hstack([np.ones((m,1)), x, x**2, x**3, x**4, x**5])
```

When you have found the answers for θ for each of the different λ values, you can check the L2-norm of theta with those provided below to confirm that your answers are correct. In numpy, you can calculate the L2-norm of a vector `x` using the command `np.linalg.norm(x)`. You should get the following norms:

	$\lambda = 0$	$\lambda = 1$	$\lambda = 10$
L2-norm(θ)	9.8894	0.85034	0.59296

Report the plot the polynomial fit for each value of λ. You will get plots similar to these:



In addition to the three λ values above, choose three additional values and learn a model for them.

2.4 Part A: Deliverables

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

1. Plot of raw data
2. For each different λ value (3 required, 3 you choose), report:
   (a) What value of θ did you get?
   (b) What is the L2-norm of the θ value you got?
   (c) A plot of the learned function. These could all be on the plot.
   (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)
3. Answer the following questions:
   (a) How the regularization parameter λ affects your model?
   (b) What would our model look like as $\lambda \to \infty$?
   (c) What λ do you think results in the model that will generalize the best?
4. Be sure to include in your `.pdf` submission your code that you wrote for this assignment.

3 Part B: Regularized logistic regression

In this part of the assignment, you will implement regularized logistic regression using Newton's Method. To begin, load the files `bx.dat` and `by.dat` into your program. This dataset represents the training set of a logistic regression problem with two features. To avoid confusion later, we will refer to the two input features contained in `bx.dat` as u and v. So in the `bx.dat` file, the first column of numbers represents the feature u, which you will plot on the horizontal axis, and the second feature represents v, which you will plot on the vertical axis.

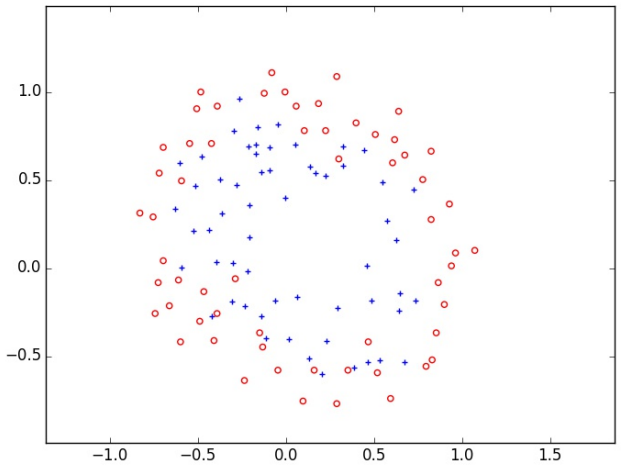
After loading the data, plot the points using different markers to distinguish between the two classifications. The commands in python/numpy will be:

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

#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,1],x[pos,2],marker='+')
plt.scatter(x[neg,1],x[neg,2],marker='o',color='r')
plt.axis('equal')
plt.show()
```

After plotting your image, it should look something like this:



We will now fit a regularized regression model to this data.

Recall that in logistic regression, the hypothesis function is

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}} = P(y = 1|x; \theta)$$

Let's look at the $\theta^T x$ parameter in the sigmoid function $g(\theta^T x)$.

In this exercise, we will assign x to be all monomials (meaning polynomial terms) of u and v up to the sixth power:

$$x = \begin{bmatrix} 1 \\ u \\ v \\ u^2 \\ uv \\ v^2 \\ u^3 \\ \vdots \\ uv^5 \\ v^6 \end{bmatrix}$$

To clarify this notation: we have made a 28-feature vector x where $x_0 = 1, x_1 = u, x_2 = v, \ldots x_{28} = v^6$. Remember that u was the first column of numbers in your `bx.dat` file and v was the second column. From now on, we will just refer to the entries of x as x_0, x_1, and so on instead of their values in terms of u and v.

To save you the trouble of enumerating all the terms of x, we've included a numpy function named `map_feature.py` that maps the original inputs to the feature vector. This function works for a single training example as well as for an entire training. To use this function, place map_feature.py in your working directory and place the following import statement at the top of file

```
from map_feature import *
```

To use the `map_feature` function, simply type

```
x = map_feature(u,v)
```

This assumes that the two original features were stored in column vectors named 'u' and 'v.' (If you had only one training example, each column vector would be a scalar.) The function will output a new feature array stored in the variable `x`. Of course, you can use any names you'd like for the arguments and the output. Just make sure your two arguments are column vectors of the same size.

Before building this model, recall that our objective is to minimize the cost function in regularized logistic regression:

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

Notice that this looks like the cost function for unregularized logistic regression, except that there is a regularization term at the beginning. We will now minimize this function using Newton's method.

3.1 Newton's method

Recall that the Newton's Method update rule is

$$\theta^{(t+1)} = \theta^{(t)} - H^{-1}\nabla_\theta J$$

This is the same rule that you used for unregularized logistic regression in Programming Assignment 1. But because you are now implementing regularization, the gradient $\nabla_\theta(J)$ and the Hessian H have different forms:

$$\nabla_\theta J = \begin{bmatrix} \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)} \\ \frac{\lambda}{m}\theta_1 + \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_1^{(i)} \\ \frac{\lambda}{m}\theta_2 + \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_2^{(i)} \\ \vdots \\ \frac{\lambda}{m}\theta_n + \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_n^{(i)} \end{bmatrix}$$

$$H = \frac{\lambda}{m}\begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} + \frac{1}{m}\left[\sum_{i=1}^{m} h_\theta(x^{(i)})(1-h_\theta(x^{(i)})x^{(i)}(x^{(i)})^\top\right]$$

Notice that if you substitute $\lambda = 0$ into these expressions, you will see the same formulas as unregularized logistic regression.

Also, remember that in these formulas,

1. $x^{(i)}$ is your feature vector, which is a 28x1 vector for this problem.
2. $\nabla_\theta J$ is a 28x1 vector.
3. $x^{(i)}(x^{(i)})^T$ and H are 28x28 matrices.
4. $y^{(i)}$ and $h_\theta(x^{(i)})$ are scalars.
5. The matrix following $\frac{\lambda}{m}$ in the Hessian formula is a 28x28 diagonal matrix with a zero in the upper left and ones on every other diagonal entry.

3.2 Run Newton's Method

Now run Newton's Method on this dataset using the three values of lambda below:

1. $\lambda = 0$ (this is the same case as non-regularized linear regression)
2. $\lambda = 1$
3. $\lambda = 10$

To determine whether Newton's Method has converged, it may help to print out the value of $J(\theta)$ during each iteration. $J(\theta)$ should not be decreasing at any point during Newton's Method. If it is, check that you have defined $J(\theta)$ correctly. Also check your definitions of the gradient and Hessian to make sure there are no mistakes in the regularization parts.

After convergence, use your values of theta to find the decision boundary in the classification problem. The decision boundary is defined as the line where

$$P(y = 1|x; \theta) = 0.5 \quad \Longrightarrow \quad \theta^T x = 0$$

Plotting the decision boundary here will be trickier than plotting the best-fit curve in linear regression. You will need to plot the $\theta^T x = 0$ line implicitly, by plotting a contour. This can be done by evaluating $\theta^T x$ over a grid of points representing the original u and v inputs, and then plotting the line where $\theta^T x$ evaluates to zero.

The plot implementation for numpy is given below. To get the best viewing results, use the same plotting ranges that we use here.

```
#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
X, 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_feature(u[i],v[j]),theta)

plt.clf()
```

```
plt.scatter(x[pos,1],x[pos,2],marker='+')
plt.scatter(x[neg,1],x[neg,2],facecolors='none',marker='o',color='r')
plt.axis('equal')
plt.contour(X,Y,Z, 0,linewidth=2)
plt.show()
```

When you are finished, your plots for the three values of λ should look similar to the ones below.

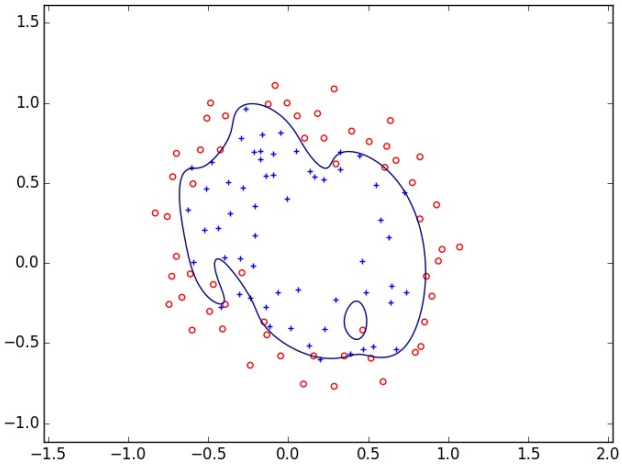


Figure 1: $\lambda = 0.0$

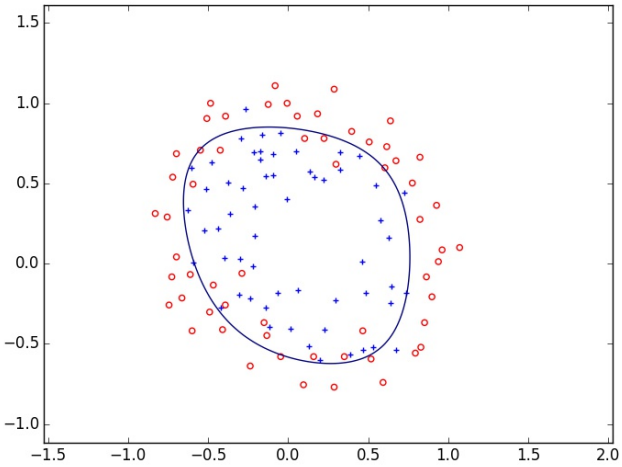


Figure 2: $\lambda = 1.0$

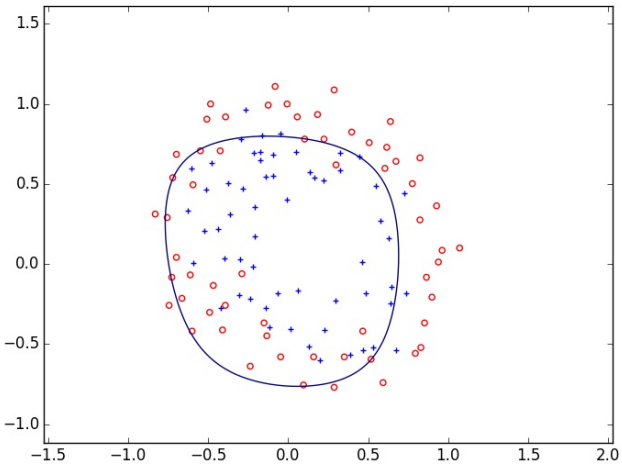


Figure 3: $\lambda = 10.0$

Finally, because there are 28 elements θ, we will not provide an element-by-element comparison for you to check. Instead, calculate the L2-norm of θ, and check it against the following norms. In numpy, you can calculate the L2-norm

of a vector `x` using the command `np.linalg.norm(x)`.

	$\lambda = 0$	$\lambda = 1$	$\lambda = 10$
L2-norm(θ)	7301.0976	4.2457	0.9401

In addition to learning a model with these λ values, please choose three additional values and learn a model for them as well.

3.3 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 $(0.5, 0.5)$?
3. What happens to the decision boundary as λ is increased?
4. What value of λ appears to give the best decision boundary (in terms of generalization)?
5. Be sure to include in your `.pdf` submission your code that you wrote for this assignment.