National University of Computer and Emerging Sciences, Lahore Campus

Course: Program: Assessment Tool	Applied Machine Learning BS (Electrical Engineering) Programming Assignment # 3	Course Code: Semester:	EE 437 Fall 2020
Total Marks:	60		

DEADLINE: Friday, November 25, 2020. **[CLO 3]**

In this assignment, you will implement logistic regression and get a good understanding of the key components of logistic regression (many other machine learning algorithms as well):

- hypothesis function
- cost function
- decision boundary
- gradient decent algorithm
- gradient checking

You will also apply your implemented logistic regression model on two small datasets and predict whether a student will be admitted to a university and a microchip should be accepted or not. These datasets will allow you to visualize the data and debug more easily.

Exercise 1

Logistic Regression

In this part of the exercise, you will build a logistic regression model to predict whether a student gets admitted into a university. Suppose that you are the administrator of a university department and you want to determine each applicant's chance of admission based on their results on two exams. You have historical data from previous applicants that you can use as a training set for logistic regression. For each training example, you have the applicant's scores on two exams and the admissions decision. Your task is to build a classification model that estimates an applicant's probability of admission based the scores from those two exams.

Visualizing the data

Before starting to implement any learning algorithm, it is always good to visualize the data if possible. Write code in Python to display a figure like Figure 1, where the axes are the two exam scores, and the positive and negative examples are shown with different markers.

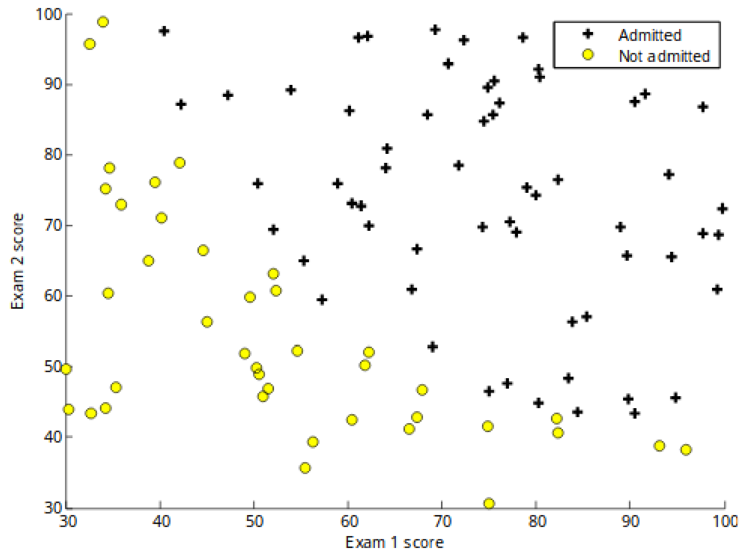

Figure 1: Scatter plot of training data

Sigmoid function

Before you start with the actual cost function, recall that the logistic regression hypothesis is defined as:

$$h_\theta(x) = g(\theta^T x),$$

where function g is the sigmoid function. The sigmoid function is defined as:

$$g(z) = \frac{1}{1+e^{-z}}.$$

Your first step is to implement the sigmoid function so it can be called by the rest of your program. When you are finished, try testing a few values by calling sigmoid(x) at the python command line. For large positive values of x, the sigmoid should be close to 1, while for large negative values, the sigmoid should be close to 0. Evaluating sigmoid(0) should give you exactly 0.5. Your code should also work with vectors and matrices. For a matrix, your function should perform the sigmoid function on every element.

Cost function and gradient

Now you will implement the cost function and gradient for logistic regression. Create a function named `costFunction` that return the cost and gradient. Recall that the cost function in logistic regression is

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right],$$

and the gradient of the cost is a vector θ where the j^{th} element (for j=0,1, … , *n*) is defined as follows:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Note that while this gradient looks identical to the linear regression gradient, the formula is actually different because linear and logistic regression have different definitions of $h_\theta(x)$.

Implement the gradient descent and find optimal parameters. On optimal parameters you should see the cost is about 0.203. Use final theta value to plot the decision boundary on the training data, resulting in a figure similar to Figure 2.

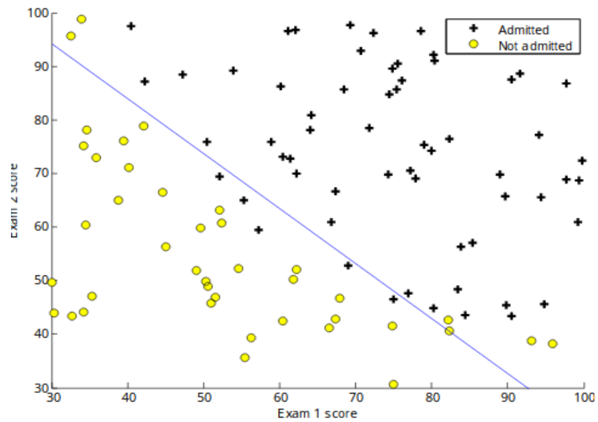

Figure 2: Training data with decision boundary

Evaluating logistic regression
After learning the parameters, you can use the model to predict whether a particular student will be admitted. For a student with an Exam 1 score of 45 and an Exam 2 score of 85, you should expect to see an admission probability of 0.776.

Exercise 2

Regularized logistic regression
In this part of the exercise, you will implement regularized logistic regression to predict whether microchips from a fabrication plant passes quality assurance (QA). During QA, each microchip goes through various tests to ensure it is functioning correctly. Suppose you are the product manager of the factory and you have the test results for some microchips on two different tests. From these two tests,you would like to determine whether the microchips should be accepted or rejected. To help you make the decision, you have a dataset of test results on past microchips, from which you can build a logistic regression model.

Visualizing the data
Similar to the exercise 1 of this assignment, write code to generate a figure like Figure 3, where the axes are the two test scores, and the positive (y= 1, accepted) and negative (y= 0, rejected) examples are shown with different markers.

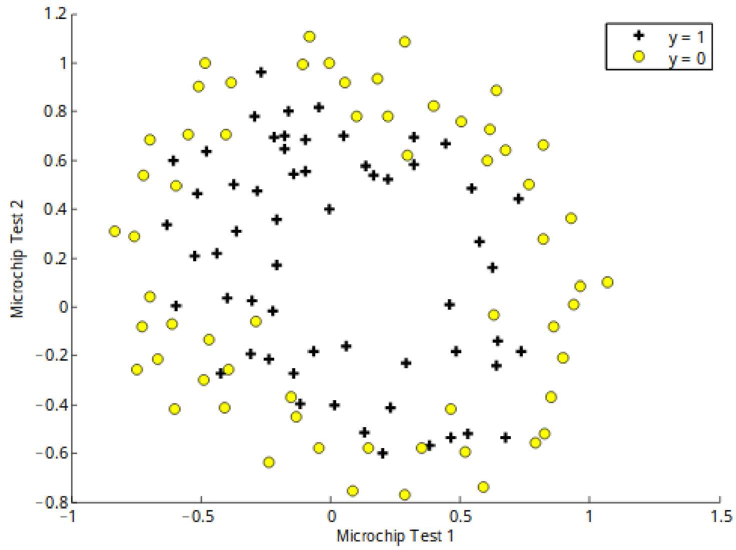

Figure 3: Plot of training data

Figure 3 shows that our dataset cannot be separated into positive and negative examples by a straight line through the plot. Therefore, a straight-forward application of logistic regression will not perform well on this dataset since logistic regression will only be able to find a linear decision boundary.

Feature mapping

One way to fit the data better is to create more features from each data point. In the next part of the exercise you will map the features into all polynomial terms of $x^1$and x^2 up to the sixth power.

$$\text{mapFeature}(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \\ x_1^3 \\ \vdots \\ x_1 x_2^5 \\ x_2^6 \end{bmatrix}$$

As a result of this mapping, our vector of two features (the scores on two QA tests) has been transformed into a 28-dimensional vector. A logistic regression classifier trained on this higher dimension feature vector will have a more complex decision boundary and will appear nonlinear when drawn in our 2-dimensional plot. While the feature mapping allows us to build a more expressive classifier, it is also more susceptible to overfitting. In the next parts of the exercise, you will implement regularized logistic regression to fit the data and also see for yourself how regularization can help combat the overfitting problem.

Cost function and gradient

Now you will implement code to compute the cost function and gradient for regularized logistic regression. Recall that the regularized cost function in logistic regression is

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

Note that you should not regularize the parameter θ_0; thus, the final summation above is for $j = 1$ to n, not $j = 0$ to n. The gradient of the cost function is a vector where the j^{th} element is defined as follows:

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \qquad \text{for } j = 0$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\left(\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \lambda\theta_j\right) \qquad \text{for } j \geq 1$$

Once you are done, call your cost function using the initial value of θ (initialized to all zeros). You should see that the cost is about 0.693.

Learning Parameters and Plotting

Implement the gradient descent and find optimal parameters. Use final theta value to plot the decision boundary on the training data, resulting in a figure similar to Figure 4.

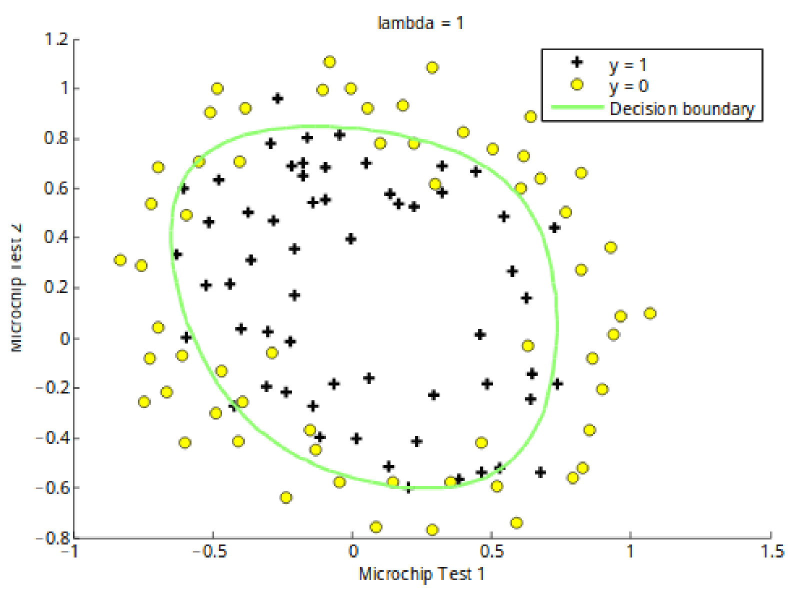

Figure 4: Training data with decision boundary (λ= 1)

Optional (Bonus exercises)

In this part of the exercise, you will get to try out different regularization parameters for the dataset to understand how regularization prevents overfitting. Notice the changes in the decision boundary as you vary λ. With a small λ, you should find that the classifier gets almost every training example correct, but draws a very complicated boundary, thus overfitting the data (Figure 5). This is not a good decision boundary: for example, it predicts that a point at x= (−0.25,1.5) is accepted (y= 1), which seems to be an incorrect decision given the training set.

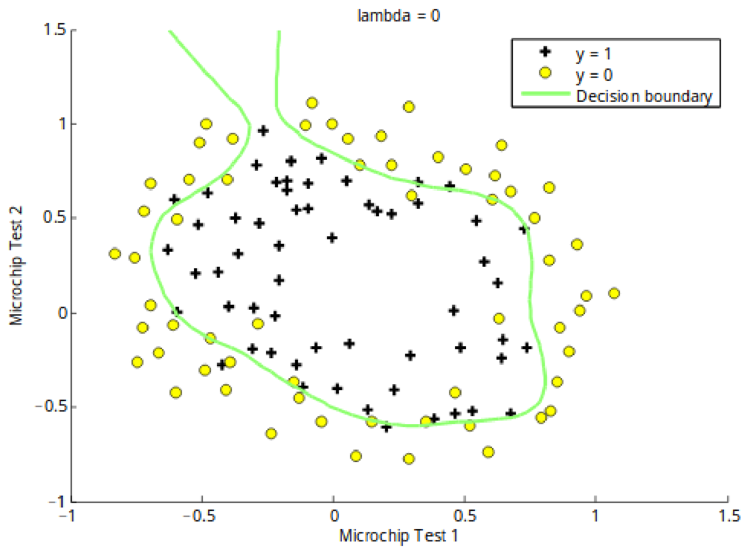

Figure 5: No regularization (Overfitting) (λ= 0)

With a larger λ, you should see a plot that shows an simpler decision boundary which still separates the positives and negatives fairly well. However, if λ is set to too high a value, you will not get a good fit and the decision boundary will not follow the data so well, thus underfitting the data (Figure6).

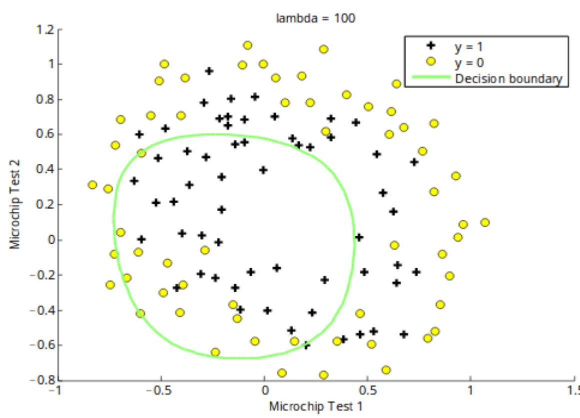

Figure 6: Too much regularization (Underfitting) (λ= 100)