# Lab#5: Regularization

Prepared by: Teeradaj Racharak (r.teeradaj@gmail.com)

## Regularized Logistic Regression[1]

In this part of exercise, we will implement regularized logistic regression to predict whether microchips from a fabrication plat 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.

**Problem 1.1 [Python from scratch].** To gain more understanding on the dataset, you are asked to complete the code for plotting the dataset. That is, you should make a 2D-plot 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 (*cf.* Figure 1).

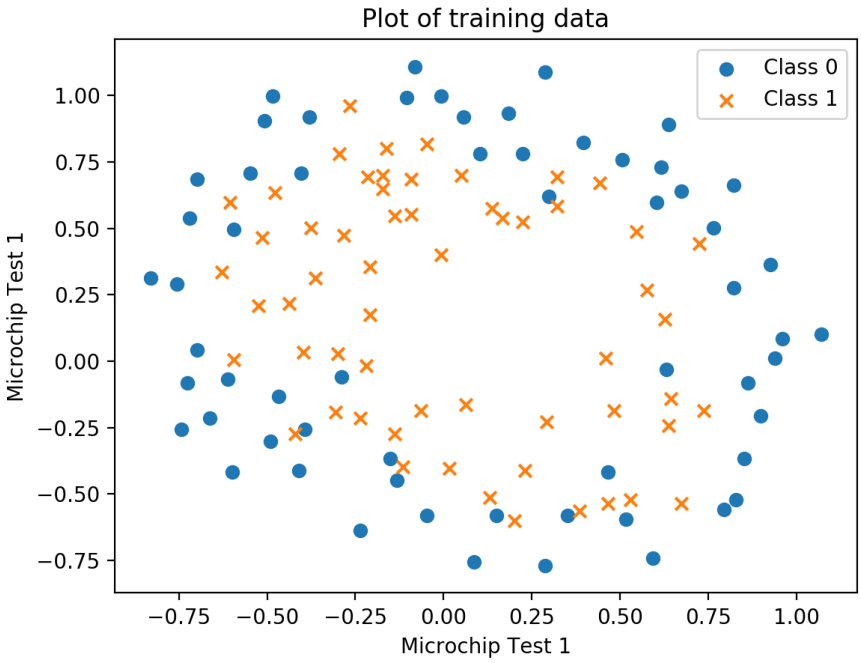


**Figure 1** Plot of training data

[1] This exercise is taken from and revised from Andrew Ng's machine learning assignments (Coursera).

**Problem 1.2 [Python with scikit-learn].** Figure 1 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 a logistic regression will not perform well on this dataset since logistic regression will only be able to find a linear decision boundary.

One way to fit the data better is to create more features (*e.g.* more polynomial terms) from each data point. In this exercise, we will map the features into all polynomial terms of $x_1$ and $x_2$ up to the sixth power as follows:

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

As a result of this mapping, our vector of two features (*i.e.* 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.

You may use the class PolynomialFeatures in scikit-learn: **https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html** to implement this preprocessing.

**Problem 1.3 [Python from scratch].** 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:

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

You should see that the cost, at theta is $[0\ldots0]^T$, is about 0.693. Noted that its vectorized version is as follows:

$$J(\theta) = \frac{1}{m}\left( \left(\log(g(X\theta))\right)^T y + \left(\log(1 - g(X\theta))^T(1 - y)\right) \right) + \frac{\lambda}{2m}\Sigma_{j=1}^{n}\theta_j^2$$

**Problem 1.4 [Python from scratch].** After you have implemented the regularized cost function, the next step is to implement the regularized gradient of the cost function. Recall that the gradient is a vector where the $j^{th}$ element is defined as follows:

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

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

You should see that the gradient, at theta is $[0 \ldots 0]^T$, is about :
$[-0.5 \quad -0.5 \quad -0.5 \quad \ldots \quad 0.0628628 \quad 0.0628628 \quad 0.0628628]^T$.
Noted that its vectorized version of the gradient is as follows:

$$\frac{\partial J(\theta)}{\partial \theta_j} \frac{1}{m} X^T\left(g(X\theta) - y\right) + \frac{\lambda}{m}\theta_j \qquad \text{for}^2\ j \geq 1$$

**Problem 1.5 [Python with SciPy].** We'll use an optimizer from SciPy (*cf.* **https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html**) to learn the optimal parameters $\theta$. Once the parameters $\theta$ are learnt, the next task is to plot a decision boundary similar to the following figure.

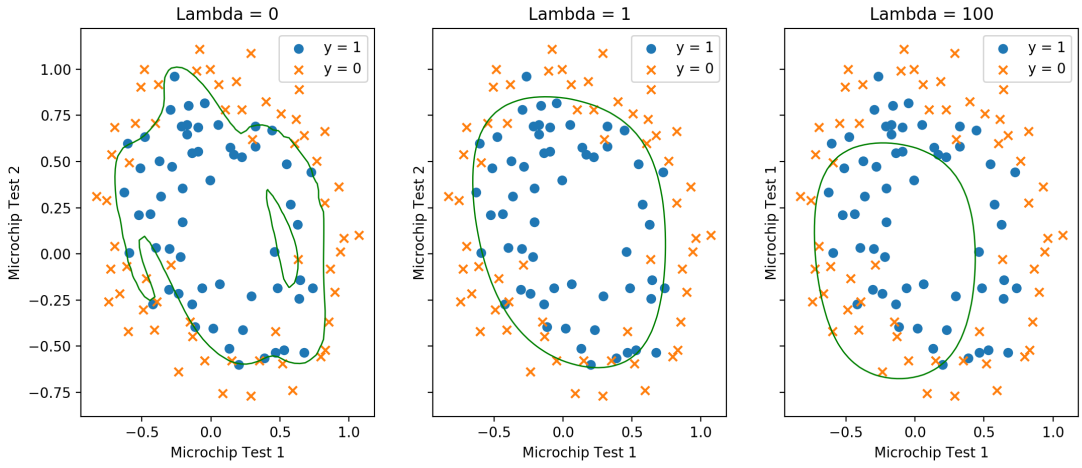


**Figure 2** Training with different regularization parameters

Notice the changes in the decision boundary as you vary $\lambda$. With a small $\lambda$, you should find that the classifier gets almost every training example correct; however, draws a very complicated boundary and overfits the data *e.g.* it will predict a point at $x = (-0.25, 1.5)$ as accepted ($y = 1$), which seems to be an incorrect decision.

[2] If $j = 0$, the regularization term is omitted.

With a larger $\lambda$, you should see a plot that shows a simpler decision boundary which still separates the positives and negatives fairly well. However, if $\lambda$ is set too high, you will not get a good fit and the decision boundary will not follow the data so well *i.e.* underfitting the data.

## Regularized Linear Regression

In this part of exercise, we will practice regularized linear regression with our previous laboratory assignment *i.e.* Problem 1 of Lab#2. To make this document shortly, please refer to Lab#2 for the description of the problem.

**Problem 2.1 [Python with TensorFlow].** Try varying the value of $\lambda$ to 10,000 and compare its change in the fitted line. Recall that the cost function and the gradient are as follows:

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

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

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

You should obtain a similar plot to the following figure.

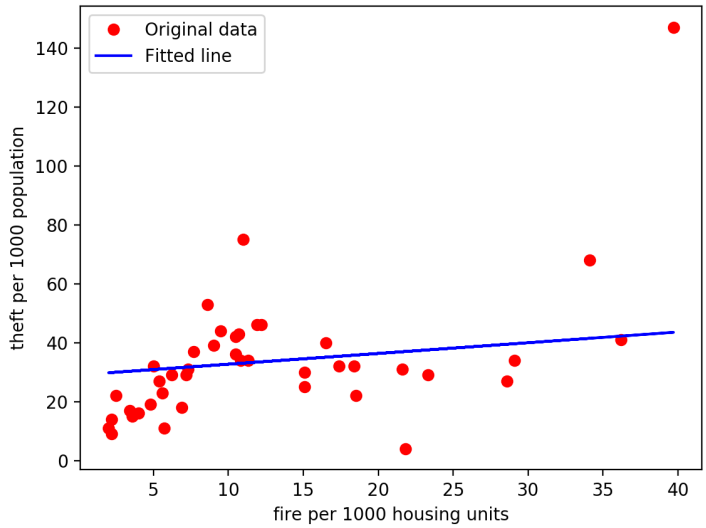


**Figure 3** Fitting with the linear regression model

**Problem 2.2 [Python from scratch].** Solve the same problem by using the normal equation instead of TensorFlow. You should obtain a similar plot to the above.