Programming Exercise 2: Logistic Regression

Machine Learning

Introduction

In this exercise, you will implement logistic regression and apply it to two different datasets. Before starting on the programming exercise, we strongly recommend watching the video lectures and completing the review questions for the associated topics.

main.py – python script the first part of this lab
main_part2.py – python script the second part of this lab
ex2data1.txt - Dataset for the first part of this lab
ex2data2.txt - Dataset for the second part of this lab

[?] *def sigmoid(z)* – Function to compute sigmoid function
[?] *def cost(theta, X, y)* - Function to compute the cost of logistic regression
[?] *def gradient(theta, X, y)* - Function to run gradient descent

? – indicates the methods you should to complete.

Where to get help

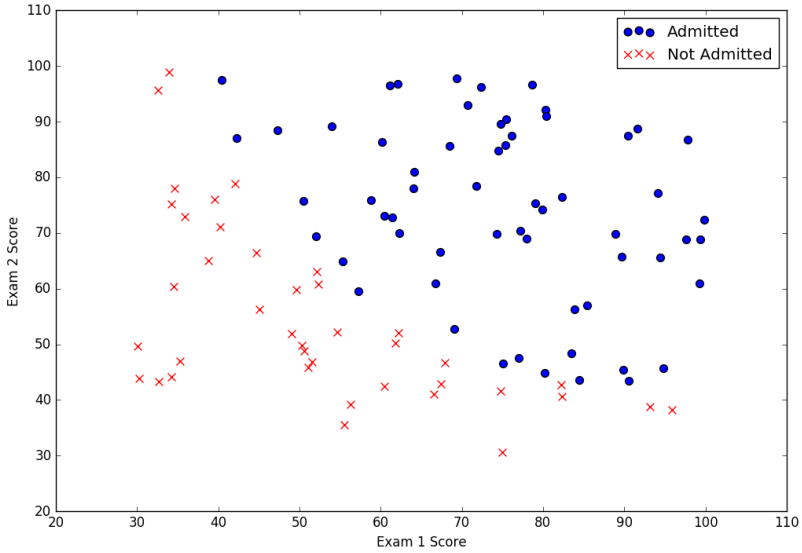
The exercises in this course use Python, a high-level programming language well-suited for numerical computations.
Python documentation can be found at https://www.python.org/ . We also strongly encourage using the online Discussions to discuss exercises with other students. However, do not look at any source code written by others or share your source code with others.

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.

Data visualization:



1.1. Implement sigmoid function

Warmup exercise: sigmoid function

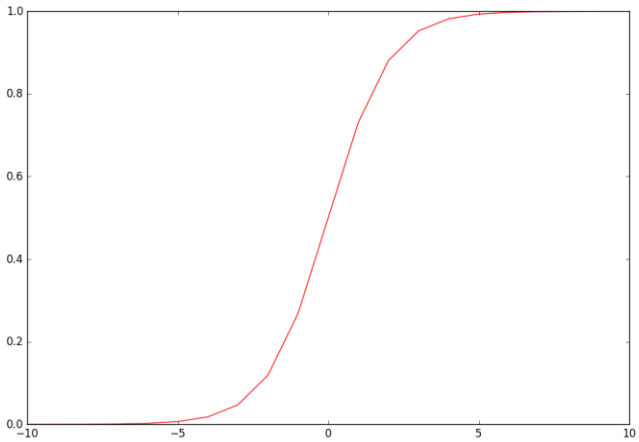
Before you start with the actual cost function, recall that the logistic regression hypothesis is defined as:

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

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

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

If you did everything correctly you should see the following plot:



1.2. Implement logistic cost function

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

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

If you did everything correctly you should see: *Cost at zero initialization:* ***0.69***

1.3. Implement gradient function

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

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 $h_\theta(x)$.

If you did everything correctly you should see: *accuracy =* ***89%***

In this exercise we use automatic function minimization method fmin_tnc which is build-in in python:

```
import scipy.optimize as opt
result = opt.fmin_tnc(func=cost, x0=theta, fprime=gradient, args=(X, y))
```

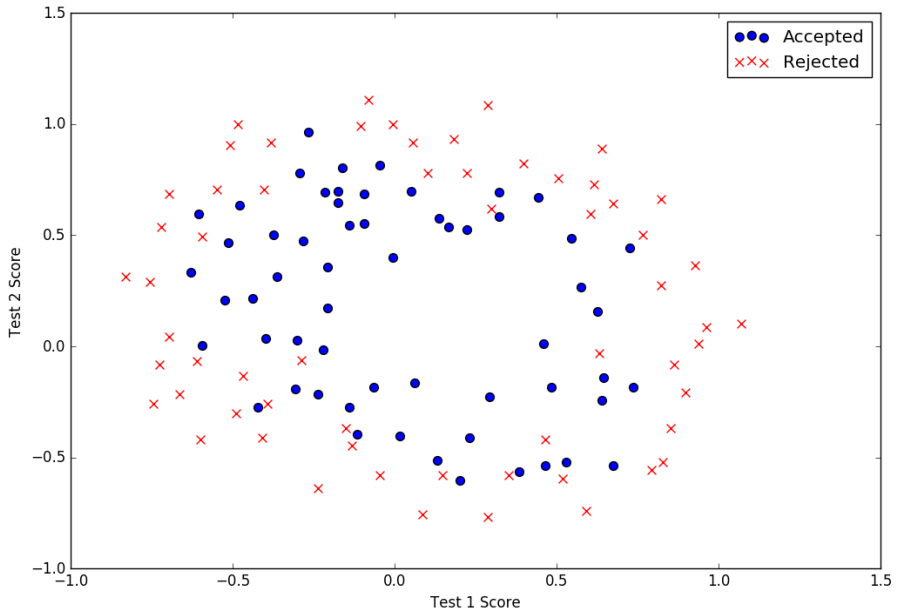
We just need to pass cost, gradient functions and initial theta values.

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.

Data visualization:



2.1. Implement Sigmoid function

Just copy-paste from the previous part ☺

2.2. Implement Regularized cost function

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

If you did everything correctly you should see: Cost at zero initialization: **0.69**

2.3. Implement Regularized Gradient function

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

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

If you did everything correctly you should see your accuracy at **77%**