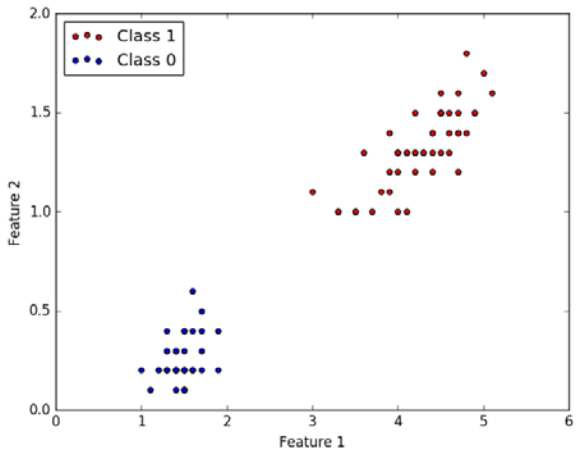
HW3: Logistic Regression

In this exercise we will explore binary classification using a logistic regression learner. There are two text files, one containing multiple features (X-Classification.txt), and the other containing labels (y-Classifications.txt). These two files will be used for training and testing. You are to use these data files for the following exercises. To get you started I have provided a PyDev Module shell with some useful hints. You do not have to use this module if you prefer to do your own scratch implementation. Hand-in all code along with a pdf document with the answers and graphs to the questions below. All submitted as a zip file.

1. Implementing Gradient Descent for Logistic Regression

a. Take a look at your features in a scatter plot, color by the class that each feature tuple represents. Provide a scatter plot of your data. It should look something like the plot below but not exactly. What can you say about your data from the plot?



b. You will now implement the gradient decent algorithm in Python for Logistic Regression. Recall the gradient descent algorithm is as follows:

$Repeat\ until\ \min_\theta J(\theta)\ is\ reached\ \{this\ is\ when\ J(\theta) <$ threshold$\}$

{

$$\theta_j := \theta_j - \alpha \left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right]$$

}

$$where\ h_\theta(x^{(i)}) = \frac{1}{1+e^{-\theta^T x^{(i)}}}$$

Evaluating the cost function will tell you when you have reached the global minimum. So we will go for as many iterations as necessary to reach the global minimum. If you recall for logistic regression the cost function is:

HW3: Logistic Regression

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

Note that the implementation of gradient descent and the summation in the cost function can be simplified by performing all these iterations simultaneously using linear algebra. This removes the need for a few loops, simplifies the code, and is much more efficient. You are welcome to implement gradient descent either way. Report your hypothesis function ($h_\theta(x) = ?$)

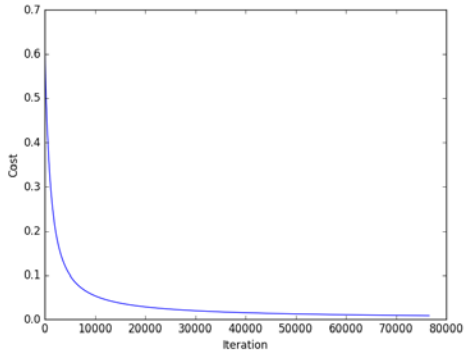
c. For evaluation you will evaluate your hypothesis $h_\theta(x)$ function on the same data you trained your learner on. Remember that your hypothesis $h_\theta(x)$ function now produces a probability $p(y = 1 \mid x;\ \theta)$. To turn this probability into a classification we define our prediction as

$$\hat{y} = \begin{cases} 0 \ if \ \hat{p} < 0.5 \\ 1 \ if \ \hat{p} \geq 0.5 \end{cases}$$

Try two different learning rates and three different cost thresholds. So you should report 6 different performance results. Make one learning rate 0.01 and one threshold 1e-7. The others you can select. Explain what you think the effect of the learning rate and cost threshold is on learning and accuracy? Also, do you think these performance measurements are overstated or understated? Why?

Learning Rate	Cost Threshold	Accuracy

d. Now enhance your code to track the cost at each step (iteration) of gradient descent and graph it. Provide the graph. It should look something like this but not exactly.



2. Implement the Scikit-Learn Logistic Regression Function

a. Now use the Scikit-Learn Logistic Regression function.
   i. Report your hypothesis function ($h_\theta(x) = ?$)
   ii. Report your accuracy predicting on the training set.
   iii. Why do you think your θ parameters and performance are different or the same, whichever is the case?