Programming Exercise 2: Logistic Regression

1 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. To get started with the exercise, you will need to download the starter code and unzip its contents to the directory where you wish to complete the exercise. Files included in this exercise

- ex2.py - Script that will help step you through the exercise
- ex2 reg.py - Script for the later parts of the exercise
- ex2data1.txt - Training set for the first half of the exercise
- ex2data2.txt - Training set for the second half of the exercise
- mapFeature.py - Function to generate polynomial features
- plotDecisionBounday.py - Function to plot classier's decision boundary
- plotData.py - Function to plot 2D classication data
- sigmoid.py - Sigmoid Function
- costFunction.py - Logistic Regression Cost Function
- predict.py - Logistic Regression Prediction Function
- costFunctionReg.py - Regularized Logistic Regression Cost

Throughout the exercise, you will be using the scripts ex2.py and ex2 reg.py. These scripts set up the dataset for the problems and make calls to functions that you will write. You do not need to modify either of them. You are only required to modify functions in other files, by following the instructions in this assignment.

1.1 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 classication model that estimates an applicant's probability of admission based the scores from those two exams. This outline and the framework code in ex2.py will guide you through the exercise.

1.2 Visualizing the data

Before starting to implement any learning algorithm, it is always good to visualize the data if possible. In the

first part of ex2.py, the code will load the data and display it on a 2-dimensional plot by calling the function plotData. You will now complete the code in plotData so that it displays a

figure like Figure 1, where the axes are the two exam scores, and the positive and negative examples are shown with different markers.

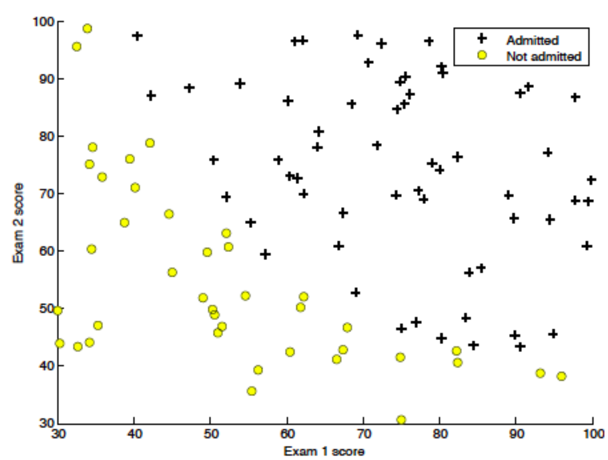

Figure 1: Scatter plot of training data

To help you get more familiar with plotting, we have left plotData.py empty so you can try to implement it yourself. However, this is an optional (ungraded) exercise. We also provide our implementation below so you can copy it or refer to it.

1.3 Implementation

1.2.1 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:

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

Your first step is to implement this function in sigmoid.py so it can be called by the rest of your program. When you are finished, try testing a few values by calling sigmoid(x). 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.

1.4 Cost function and gradient

Now you will implement the cost function and gradient for logistic regression. Complete the code in costFunction.py to return the cost and gradient. Recall that the cost function in logistic regression is

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

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

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

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 $h_\theta(x)$. Once you are done, ex2.py will call your costFunction using the initial parameters of θ. You should see that the cost is about 0.693. You should now submit the cost function and gradient for logistic regression. Make two submissions: one for the cost function and one for the gradient. This final value will then be used to plot the decision boundary on the training data, resulting in a figure similar to Figure 2. We also encourage you to look at the code in plotDecisionBoundary.py to see how to plot such a boundary using the values.

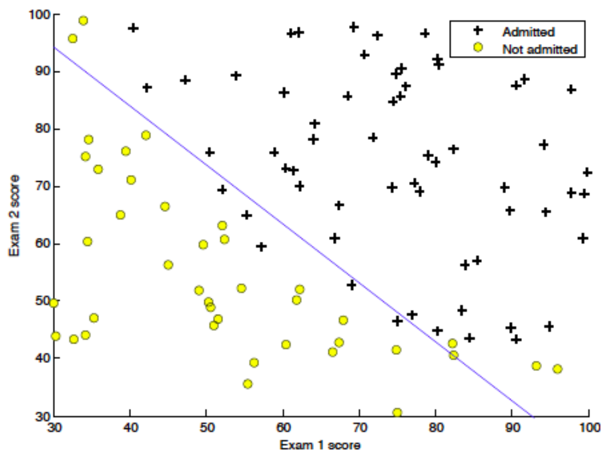

Figure 2: Training data with decision boundary

1.5 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. Another way to evaluate the quality of the parameters we have found is to see how well the learned model predicts on our training set. In this part, your task is to complete the code in predict.py. The predict function will produce 1 or 0 predictions given a dataset and a learned parameter vector . After you have completed the code in predict.py, the ex2.py script will proceed to report the training accuracy of your classi er by computing the percentage of examples it got correct. You should now submit the prediction function for logistic regression. 7

1.6 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. You will use another script, ex2 reg.py to complete this portion of the exercise.

1.7 Visualizing the data

Similar to the previous parts of this exercise, plotData is used 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

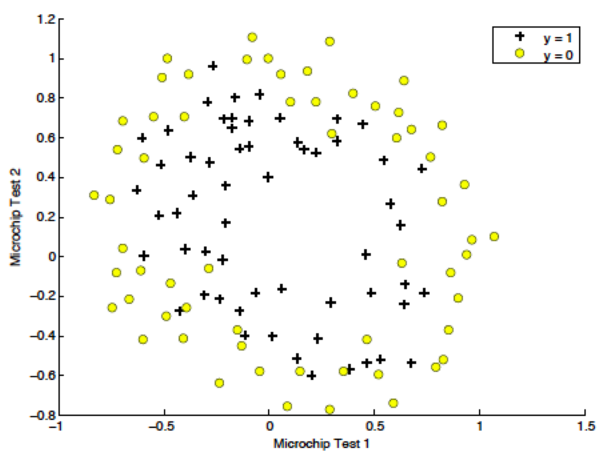

Figure 3: Plot of training data

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

nd a linear decision boundary.

2 Feature mapping

One way to

t the data better is to create more features from each data point. In the provided function mapFeature.py, we will map the features into all polynomial terms of x1 and x2 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 classier 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 classier, it 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.

2.1 Cost function and gradient

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

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

Note that you should not regularize the parameter 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:

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

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\left(\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)} + \lambda\theta_j\right), \text{ for j} = 1,\ 2,\ 3,\ ...$$

Once you are done, ex2 reg.py will call your costFunctionReg function using the initial value of (initialized to all zeros). You should see that the cost is about 0.693. You should now submit the cost function and gradient for regularized logistic regression. Make two submissions, one for the cost function and one for the gradient.

3 Theoretical Questions

- Prove that the negative log likelihood of observing $\left(\left(x^{(1)}, y^{(1)}\right), \cdots, \left(x^{(m)}, y^{(m)}\right)\right)$ with a logistic regression with parameter θ equals the following:

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

  Furthermore prove that the gradients are computed in the following way:

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

  where $h_\theta(x) = p(y=1|x) = \frac{1}{1+e^{-x^t\theta}}$.

- In this question you will prove that the regularized cost function is equivalent to working with the posterior. Prove that the negative log likelihood of observing $\left(\left(x^{(1)}, x^{(1)}\right), \cdots, \left(x^{(m)}, x^{(m)}\right)\right)$ with a logistic regression with parameter θ

and a Gaussian prior on θ with mean 0 and covariance matrix $\frac{1}{\sqrt{\lambda}}$I (or in other words $\theta \sim$ N(0, $\frac{1}{\sqrt{\lambda}}$I)) equals:

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

Furthermore prove that gradients are computed in the following way:

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

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\sum_{i=1}^{m} \left(h_\theta \left(x^{(i)} \right) - y^{(i)} \right) x_j^{(i)} + \lambda \theta_j \right), \text{ for j} = 1, 2, 3, ...$$

where $h_\theta(x) = p(y = 1|x) = \frac{1}{1+e^{-x^t \theta}}$.