Lab 3
Logistic Regression & Classification

1. Introduction

In this lab, you will implement logistic regression and apply it to a specific dataset. Before starting on the exercise, we strongly recommend looking at the course notes form Lecture 5.

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. If needed, use the cd command in MATLAB to change to this directory before starting this exercise.

Files included in this exercise:

- Lab3.m - MATLAB script that steps you through the exercise
- Lab3_data1.txt - Training set for the first half of the exercise
- plotDecisionBoundary.m - Function to plot classifier's decision boundary

Other functions, that you will need to complete:

- plotData.m - Function to plot 2D classification data
- sigmoid.m - Sigmoid Function
- costFunction.m - Logistic Regression Cost Function
- predict.m - Logistic Regression Prediction Function

Throughout the exercise, you will be using the script Lab3.m. This script sets up the dataset for the problem and makes calls to functions that you will write. You do not need to modify this script. You are only required to modify functions in other files, by following the instructions in this assignment.

2. 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.

2.1 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 Lab3.m, 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.

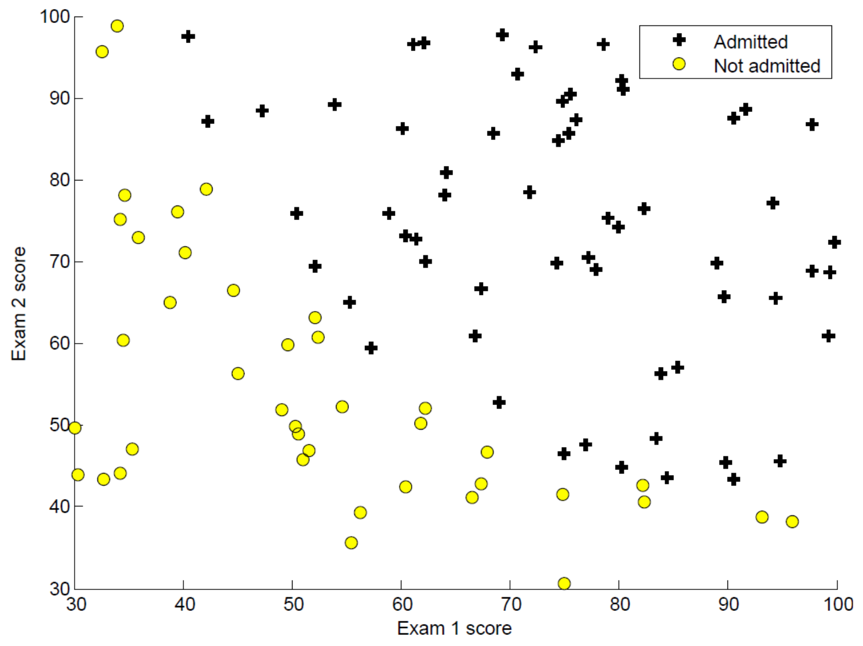

Figure 1: Scatter plot of training data

```
% Find Indices of Positive and Negative Examples
pos = find(y==1); neg = find(y == 0);

% Plot Examples
plot(X(pos, 1), X(pos, 2), 'k+','LineWidth', 2, ...
     'MarkerSize', 7);
plot(X(neg, 1), X(neg, 2), 'ko', 'MarkerFaceColor', 'y', ...
     'MarkerSize', 7);
```

To help you get more familiar with plotting, we have left plotData.m empty so you can try to implement it yourself. However, this is an *optional exercise*. We also provide our implementation so you can copy it or refer to it. If you choose to copy our example, make sure you learn what each of its commands is doing by consulting the MATLAB documentation.

2.2 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.m* 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 MATLAB 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.**
Hint: use . in front of operators to get element-wise operation performed!

2.3 Cost function and gradient

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

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

and the gradient of the cost is a vector of the same length as ϑ where the jth element (for $j = 0, 1, \ldots, n$) 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 $h_\vartheta(x)$.
Once you are done, Lab3.m will call your *costFunction* using the initial parameters of ϑ.

You should see that the cost is about 0.693.

For the initial gradient, you should get a vector about (-0.1, -12, -11.2)

2.4 Learning parameters using fminunc

In the previous assignment, you found the optimal parameters of a linear regression model by implementing gradent descent. You wrote a cost function and calculated its gradient, then took a gradient descent step accordingly. This time, instead of taking gradient descent steps, you will use an MATLAB built-in function called *fminunc*.

MATLAB's *fminunc* is an **optimization solver** that finds the minimum of an unconstrained function. *Constraints in optimization often refer to constraints on the parameters, for example, constraints that bound the possible values ϑ can take (e.g., ϑ 1). Logistic regression does not have such constraints since ϑ is allowed to take any real value.*
For logistic regression, you want to optimize the cost function $J(\vartheta)$ with parameters ϑ.

Concretely, you are going to use fminunc to find the best parameters ϑ for the logistic regression cost function, given a fixed dataset (of X and y values). You will pass to fminunc the following inputs:

- The initial values of the parameters we are trying to optimize.
- A function that, when given the training set and a particular ϑ, computes the logistic regression cost and gradient with respect to ϑ for the dataset (X, y)

In Lab3.m, we already have code written to call *fminunc* with the correct arguments.

```
%  Set options for fminunc
options = optimset('GradObj', 'on', 'MaxIter', 400);

%  Run fminunc to obtain the optimal theta
%  This function will return theta and the cost
[theta, cost] = ...
    fminunc(@(t)(costFunction(t, X, y)), initial-theta, options);
```

In this code snippet, we first defined the options to be used with fminunc. Specifically, we set the *GradObj* option to *on*, which tells *fminunc* that our function returns both the cost and the gradient. This allows *fminunc* to use the gradient when minimizing the function. Furthermore, we set the *MaxIter* option to 400, so that *fminunc* will run for at most 400 steps before it terminates.

To specify the actual function we are minimizing, we use a "short-hand" for specifying functions with the *@(t) (costFunction(t, X, y)*) . This creates a function, with argument t, which calls your *costFunction*. This allows us to wrap the *costFunction* for use with *fminunc*.

If you have completed the *costFunction* correctly, *fminunc* will converge on the right optimization parameters and return the final values of the cost and ϑ. Notice that by using *fminunc*, you did not have to write any loops yourself, or set a learning rate like you did for gradient descent. This is all done by *fminunc*: **you only needed to provide a function calculating the cost and the gradient.**

Once *fminunc* completes, Lab3.m will call your *costFunction* function using the optimal parameters of ϑ. You should see that the cost is about 0.203.

This final ϑ 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.m* to see how to plot such a boundary using the ϑ values.

2.5 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.

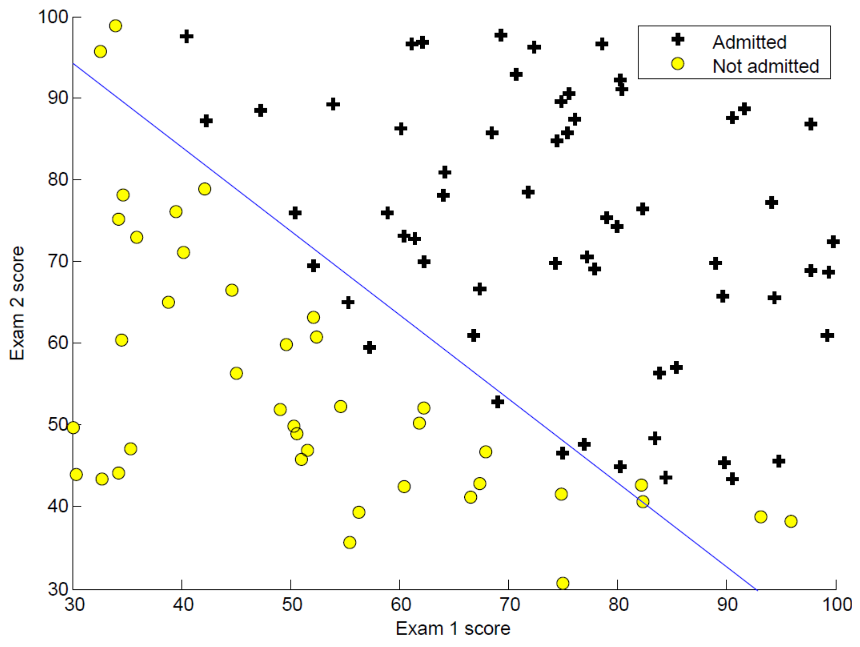

Figure 2: Training data with decision boundary

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.m*. The predict function will produce “1” or “0” predictions given a dataset and a learned parameter vector ϑ.

After you have completed the code in *predict.m*, the Lab3.m script will proceed to report the training accuracy of your classifier by computing the percentage of examples it got correct.

3. Additional exercises

a) Predict the admission result for 3 additional students. Highlight the admission probability as well.

b) Plot the predicted values with a different color and shape on the same plot in Figure 2.

c) Apply gradient descent iteratively, as done in Lab 2. Set the learning rate to 0.01.

d) Plot the decision boundary line for each iteration at exercise 3c). Use the same figure as before.

e) Change the learning rate (either increase it or decrease it by a factor of 10). Apply gradient descent and plot the decision boundary line on a new figure.