# Programming Exercise 3: Logistic Regression

In this exercise, you will implement logistic regression and apply it to a dataset.

The file `ex1data2.txt` contains a training set of two exams and the admission results of students. The first column is the score of exam 1 and the second column is the score for exam 2. Third column is the admission decision of the student (1 means accepted, 0 implies denied).

For working on this exercise, you will be given a starter code. You may need to use `cd` command to change to the right directory.

Following files are either complete or partially complete. If it is partially complete, you will have to complete it.

| | |
|---|---|
| `ex2.m` | `%%complete` |
| `ex2data1.txt` | `Dataset` |
| `plotData.m` | `Hint Given` |
| `Sigmoid.m` | `%incomplete` |
| `costFunction.m` | `%incomplete` |
| `predict.m` | `%incomplete` |
| | |

You will be running `ex2.m`. The code in `ex2.m` will set up the dataset for the problem and make calls to functions that you will have to write. This file is almost complete. However, you will have to modify functions in other files according to the instructions that will be provided later in this document.

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

### 1.1 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 `ex2.m`, the code will load the data and display it on a 2-dimensional plot by calling the function `plotData`.

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.

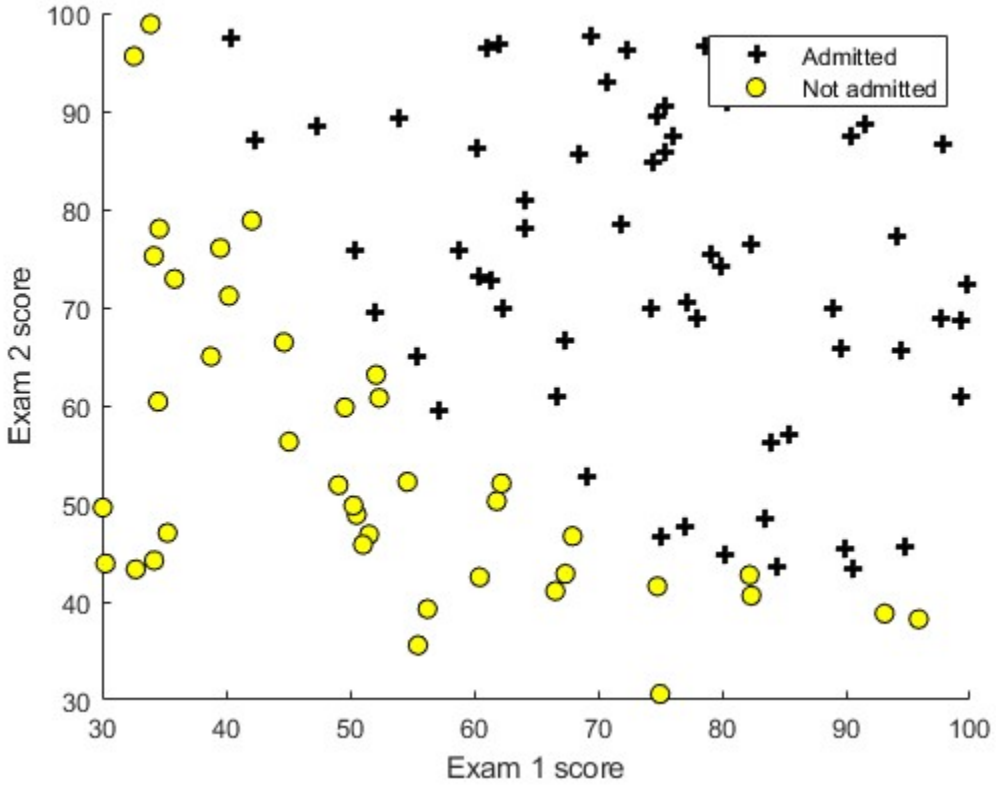



**Figure 1**

```
pos = find(y==1); neg = find(y == 0);
plot(X(pos, 1), X(pos, 2), 'k+','LineWidth', 2,'MarkerSize', 7);
plot(X(neg, 1), X(neg, 2), 'ko', 'MarkerFaceColor', 'y', 'MarkerSize', 7);
```

**Hint: See code above**

### 2. Implementation

### 2.1 Sigmoid 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:

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

You now have 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 Octave/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.

**2.2 Cost function and gradient**

Next you need to implement the cost function and gradient for logistic regression.
You now implement `costFunction.m` which should return the cost and gradient.
Cost function in logistic regression is different than linear regression and is written as follows:

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

Recall, for linear regression it was

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta\big(x^{(i)}\big) - y^{(i)}\,)^2$$

The gradient for the cost function is a vector of the same length as θ where the $J^{th}$ element (for j = 0,1, …., n) is defined as follows:

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

It is important to remember that even though this gradient looks identical to the linear regression gradient, the formula is different as linear and logistic regression has different definition of $h_\theta(x)$.

Once you are done, `ex2.m` will call your `costFunction` using the initial parameters of θ. You should see that the cost is about 0.693.

**<u>Gradient Descent Theory</u>**

In this part, you will fit the linear regression parameters θ to our dataset using gradient descent. Your goal is to find θ through gradient descent.

**Update equations:**

The objective of logistic regression is to minimize the cost function

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

where the hypothesis $h_\theta(x)$ is given by the model
$h_\theta(x) = g(\theta^T x)$ = $g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4 + \theta_5 x_5 + \theta_6 x_6 + \ldots.)$

Remember that the parameters of your model are the $\theta_j$ values. These are the values you will adjust to minimize cost J(θ). One way to do this is to use the batch gradient descent algorithm. In batch gradient descent, each iteration performs the update as follows:

```
repeat until convergence: {
```

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

```
}
```

However, for n features, we have repeat 'n' times.

```
repeat until convergence: {
```

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right).x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right).x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_2^{(i)}$$

...

}

(You have to update $\theta_j$ simultaneously for all $j$)

With each step of gradient descent, your parameters $\theta_j$ come closer to the optimal values that will achieve the lowest cost $J(\theta)$.

## 2.3 Learning Parameters using `fminunc`

In the previous assignment, you found the optimal parameters of a linear regression model by implementing gradient 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 Octave/MATLAB built-in function called `fminunc`.
Octave/MATLAB's `fminunc` is an optimization solver that finds the minimum of an unconstrained function. For logistic regression, you want to optimize the cost function $J(\theta)$ with parameters $\theta$.

So, you are going to use `fminunc` to find the best parameters $\theta$ 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 $\theta$, computes the logistic regression cost and gradient with respect to $\theta$ for the dataset (X, y)

In `ex2.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.

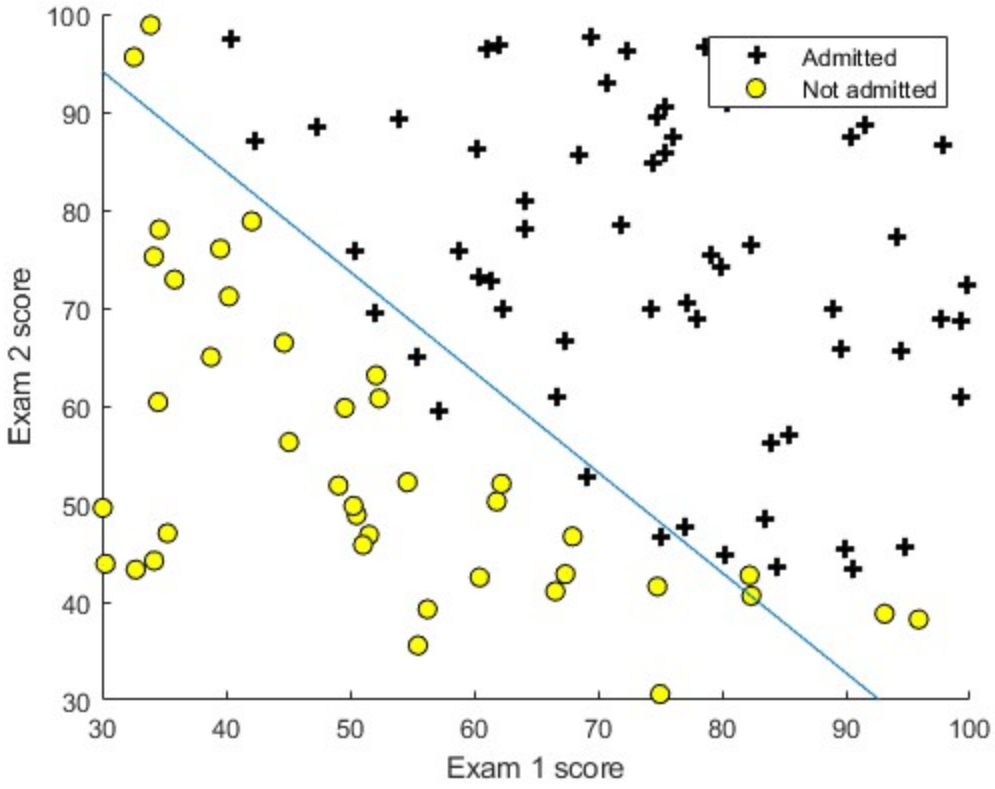
To specify the actual function we are minimizing, we use a "short-hand" for specifying functions with the `@(t) (costFunction(t, X, y) )`. This creates a function, with argument t, which calls your `costFunction`. This allows us to wrap the `costFunction` for use with `fminunc`.

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, `ex2.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 like **Figure 2**. We have provided the code for drawing the boundary line between the two classes in `plotDecisionBoundary`.m. You should look at the code in `plotDecisionBoundary.m` to see how to plot such a boundary using the θ values.

**2.3 (a): Why are we using `fminunc`? What does it do? What's the advantage of `fminunc` over gradient descent?**




### 3. 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.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 `ex2.m` script will proceed to report the training accuracy of your classifier by computing the percentage of examples it got correct.

**Distribution of points**

Finish in the class, for 100% points.
Submit by midnight today, for 92% of points.
Submit by midnight tomorrow, for 88% of points.
Submit by midnight day after tomorrow, for 84% of points.

| **Problem** | **Points** |
|---|---|
| Sigmoid | 5 |
| plotData | 5 |
| Compute cost for logistic Regression | 20 |
| Compute gradient for logistic Regression | 20 |
| Predict function | 10 |
| 2.3 (a) | 5 |