

Programming Exercise 3: Logistic Regression

Introduction

In this exercise, you will implement logistic regression and apply it to a dataset. 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 Octave/MATLAB to change to this directory before starting this exercise.

In this 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 on the scores from those two exams. This outline and the framework code in `ex3.m` will guide you through the exercise.

Files included in this exercise

`ex3.m` - Octave/MATLAB script that steps you through the exercise

`ex3data1.txt` – Training dataset for this exercise

`plotDecisionBoundary.m` - Function to plot classifier's decision boundary

[*] `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 `ex3.m`. This script sets up the dataset for the problems and makes calls to functions that you will write. You do not need to modify `ex3.m`. You are only required to modify functions in other files, by following the instructions in this assignment.

[*] indicates files you will need to complete

1 Visualizing the data and Sigmoid function

Before starting to implement any learning algorithm, it is always good to visualize the data if possible. In the first part of `ex3.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.

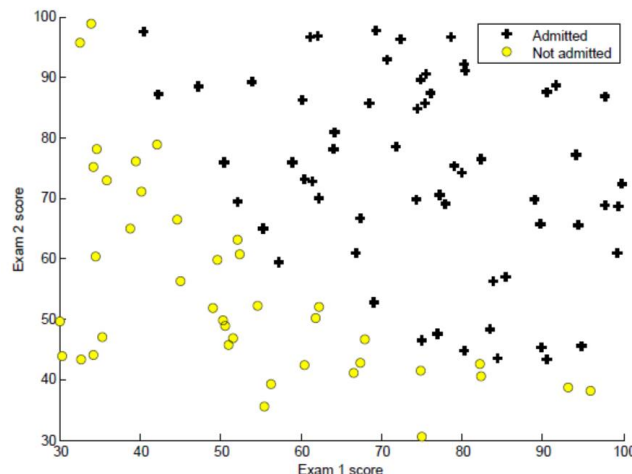


Figure 1: Scatter plot of training data

To help you get more familiar with plotting, `plotData.m` is left empty so you can try to implement it yourself. However the correct implementation is provided below so you can copy it or refer to it. If you choose to copy the given example, make sure you learn what each of its commands is doing by consulting the Octave/MATLAB documentation.

```
% 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);
```

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

$$h(x) = g(X * \theta),$$

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 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 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

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

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

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

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 $h(x)$.

Once you are done, `ex3.m` will call your `costFunction` using the initial parameters of θ . You should see that the cost is about 69.3.

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 a function. For logistic regression, you want to optimize the cost function $J(\theta)$ 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 ex3.m, there is already 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, first the options to be used with `fminunc` are defined. Specifically, the `GradObj` option is set to `on`, which tells `fminunc` that the `costFunction` function returns both the cost and the gradient. This allows `fminunc` to use the gradient when minimizing the function. Furthermore, the `MaxIter` option is set to 400, so that `fminunc` will run for at most 400 steps before it terminates.

To specify the actual function to be minimized, a “short-hand” is used for specifying functions with the `@(t) (costFunction(t, X, y))`. This creates a function, with argument `t`, which calls your `costFunction`. This will make it possible 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 (please note that this part of the program might take few seconds before it can calculate θ parameters that minimize the cost function). 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, `ex3.m` will print out the cost value calculated using the optimal parameters of θ . You should see that the cost is about 20.3.

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.

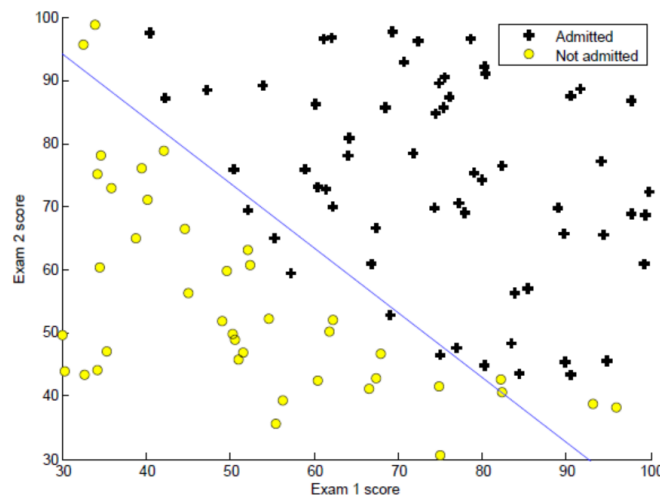


Figure 2: Training data with decision boundary

4 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 `ex3.m` script will proceed to report the training accuracy of your classifier by computing the percentage of examples it got correct.