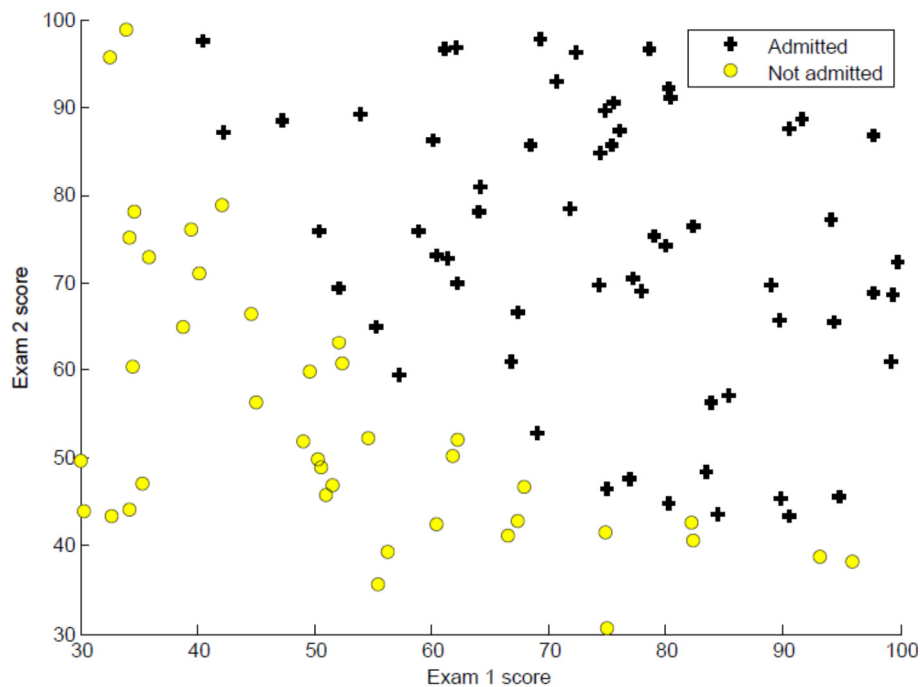


Homework 03**Logistic Regression**

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 the scores from those two exams. This outline and the framework code in **hw3.m** will guide you through the exercise.

Part 1: Visualizing the data (0 points)

Before starting to implement any learning algorithm, it is always good to visualize the data if possible. In the first part of **hw3.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 the following one, where the axes are the two exam scores, and the positive and negative examples are shown with different markers.



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 (ungraded) exercise. We also provide our implementation below 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 Octave documentation.

Homework 03

```
% 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);
```

Part 2: Sigmoid Function (10 points)

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 octave 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.

Part 3: Cost Function and Gradient (30 points)

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 j th element (for $j = 0, 1, \dots, n$) is defined as follows:

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

Homework 03

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 the hypothesis. Once you are done, **hw3.m** will call your **costFunction** using the initial parameters of θ . You should see that the cost is about **0.693**.

Part 4: Learning parameters using **fminunc (0 points)**

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 built-in function called **fminunc**. Octave'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 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)

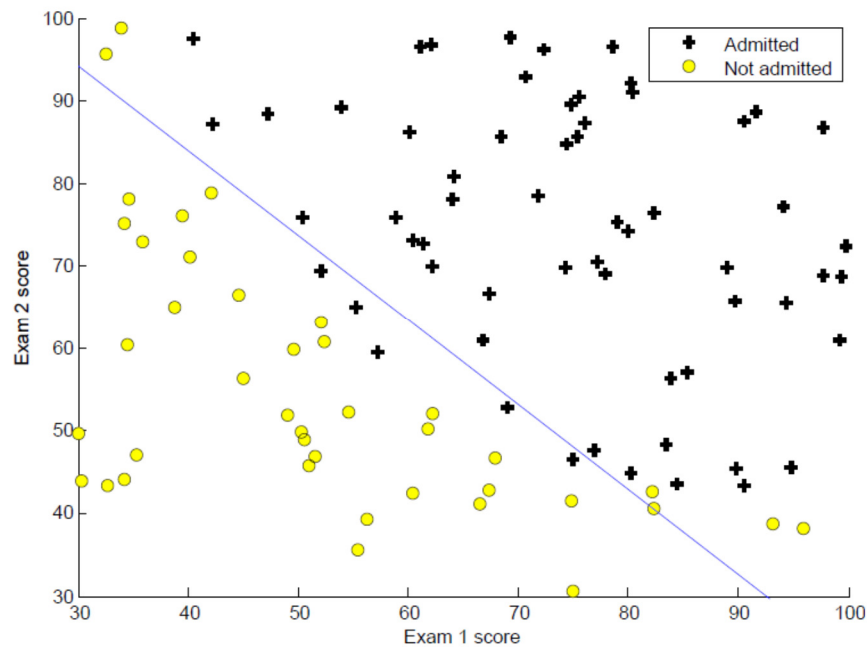
In **hw3.m**, we already have code written to call **fminunc** with the correct arguments. Specifically, we set the **GradObj** option to **on**, which tells **fminunc** that our function returns both the cost and the gradient. This allows **fminunc** to use the gradient when minimizing the function. 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, **hw3.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 the following one.

Homework 03



Part 5: Evaluating logistic regression (10 points)

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 **hw3.m** script will proceed to report the training accuracy of your classifier by computing the percentage of examples it got correct. You should expect to see a training accuracy of **89%**.

Submission:

To submit, turn in the following files on Canvas:

- **sigmoid.m**
- **costFunction.m**
- **predict.m**