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

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 $h_\theta(x)$.

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.

You should now submit your solutions.

1.2.3 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 Octave/-MATLAB built-in function called `fminunc`.

Octave/MATLAB's `fminunc` is an optimization solver that finds the minimum of an unconstrained[2] 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 `ex2.m`, we already have code written to call `fminunc` with the correct arguments.

[2] Constraints in optimization often refer to constraints on the parameters, for example, constraints that bound the possible values θ can take (e.g., $\theta \leq 1$). Logistic regression does not have such constraints since θ is allowed to take any real value.

```
%  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 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.

1.2.4 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

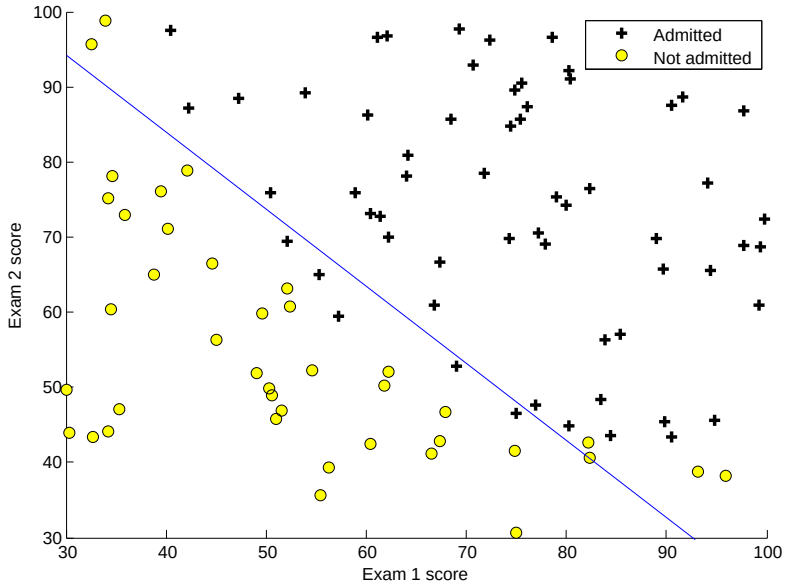

Figure 2: Training data with decision boundary

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.

You should now submit your solutions.

2 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.m` to complete this portion of the exercise.

2.1 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.

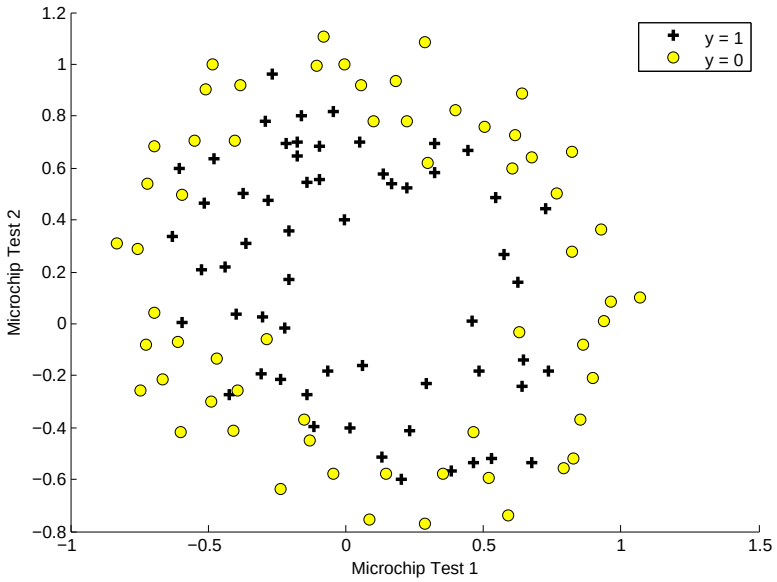

Figure 3: Plot of training data

Figure 3 shows that our dataset cannot be separated into positive and negative examples by a straight-line through the plot. Therefore, a straightforward application of logistic regression will not perform well on this dataset since logistic regression will only be able to find a linear decision boundary.

2.2 Feature mapping

One way to fit the data better is to create more features from each data point. In the provided function `mapFeature.m`, we will map the features into all polynomial terms of x_1 and x_2 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 classifier 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 classifier, 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.

2.3 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.m` to return the cost and gradient.

Recall that the regularized cost function in logistic regression is

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

Note that you should not regularize the parameter θ_0. In **Octave/MATLAB**, recall that indexing starts from 1, hence, you should not be regularizing the `theta(1)` parameter (which corresponds to θ_0) in the code. The gradient of the cost function is a vector where the j^{th} element is defined as follows:

$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \qquad \text{for } j = 0$$

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

Once you are done, `ex2_reg.m` will call your `costFunctionReg` function using the initial value of θ (initialized to all zeros). You should see that the cost is about 0.693.

You should now submit your solutions.

2.3.1 Learning parameters using `fminunc`

Similar to the previous parts, you will use `fminunc` to learn the optimal parameters θ. If you have completed the cost and gradient for regularized logistic regression (`costFunctionReg.m`) correctly, you should be able to step through the next part of `ex2_reg.m` to learn the parameters θ using `fminunc`.

2.4 Plotting the decision boundary

To help you visualize the model learned by this classifier, we have provided the function `plotDecisionBoundary.m` which plots the (non-linear) decision boundary that separates the positive and negative examples. In `plotDecisionBoundary.m`, we plot the non-linear decision boundary by computing the classifier's predictions on an evenly spaced grid and then and drew a contour plot of where the predictions change from $y = 0$ to $y = 1$.

After learning the parameters θ, the next step in `ex_reg.m` will plot a decision boundary similar to Figure 4.

2.5 Optional (ungraded) exercises

In this part of the exercise, you will get to try out different regularization parameters for the dataset to understand how regularization prevents overfitting.

Notice the changes in the decision boundary as you vary λ. With a small λ, you should find that the classifier gets almost every training example correct, but draws a very complicated boundary, thus overfitting the data (Figure 5). This is not a good decision boundary: for example, it predicts that a point at $x = (-0.25, 1.5)$ is accepted ($y = 1$), which seems to be an incorrect decision given the training set.

With a larger λ, you should see a plot that shows an simpler decision boundary which still separates the positives and negatives fairly well. However, if λ is set to too high a value, you will not get a good fit and the decision boundary will not follow the data so well, thus underfitting the data (Figure 6).

You do not need to submit any solutions for these optional (ungraded) exercises.

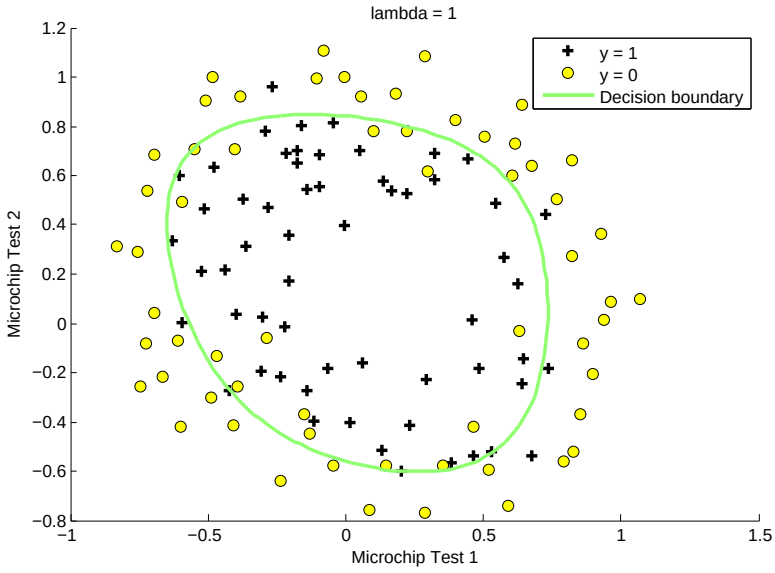

Figure 4: Training data with decision boundary ($\lambda = 1$)

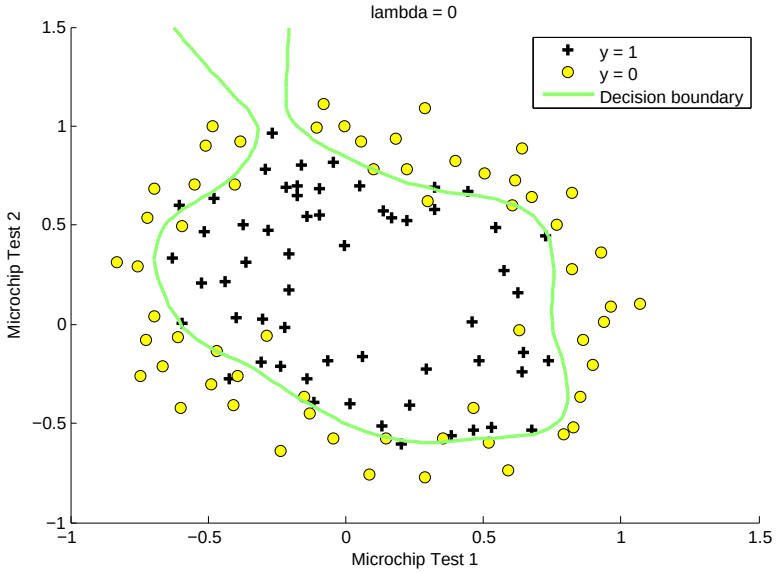

Figure 5: No regularization (Overfitting) ($\lambda = 0$)

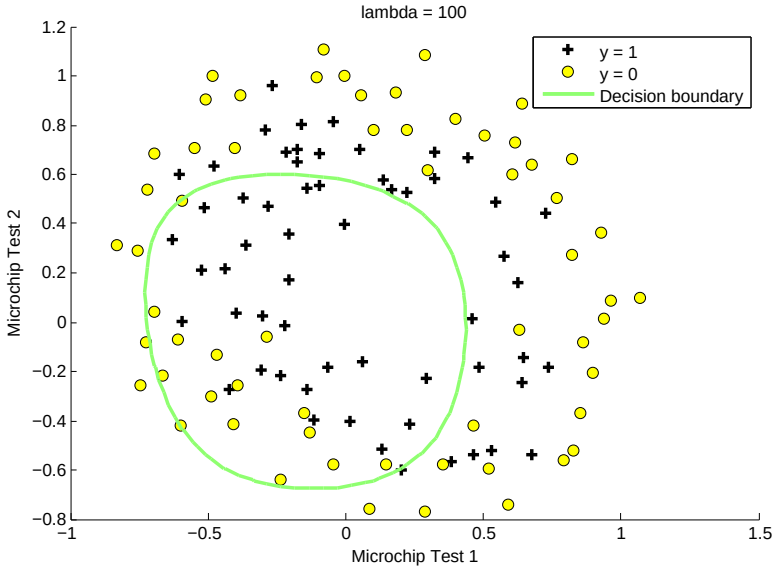

Figure 6: Too much regularization (Underfitting) ($\lambda = 100$)

Submission and Grading

After completing various parts of the assignment, be sure to use the `submit` function system to submit your solutions to our servers. The following is a breakdown of how each part of this exercise is scored.

Part	**Submitted File**	**Points**
Sigmoid Function	`sigmoid.m`	5 points
Compute cost for logistic regression	`costFunction.m`	30 points
Gradient for logistic regression	`costFunction.m`	30 points
Predict Function	`predict.m`	5 points
Compute cost for regularized LR	`costFunctionReg.m`	15 points
Gradient for regularized LR	`costFunctionReg.m`	15 points
Total Points		100 points

You are allowed to submit your solutions multiple times, and we will take only the highest score into consideration.