Programming Exercise 3: Logistic Regression

Machine Learning

Introduction

In this exercise, you will implement logistic regression and apply it to two different datasets. 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.

You can log into your CougarNet and download MATLAB from this website: https://uh.edu/software-downloads/index.php.

Files included in this exercise

ex3 reg.m - Octave/MATLAB script for the later parts of the exercise
ex3data1.txt - Training set for the first half of the exercise
ex3data2.txt - Training set for the second half of the exercise
plotData.m - Function to plot 2D classification data
mapFeature.m - Function to generate polynomial features
plotDecisionBoundary.m - Function to plot classifier's decision boundary
[y] ex3.m - Octave/MATLAB script that steps you through the exercise
[y] sigmoid.m - Sigmoid Function
[y] costFunction.m - Logistic Regression Cost Function
[y] predict.m - Logistic Regression Prediction Function
[y] costFunctionReg.m - Regularized Logistic Regression Cost
y indicates files you will need to complete

Files needed to be submit

[1] ML_ex3 – Include all the code (You need to complete ex3.m, sigmoid.m, costFunction.m, predict.m and costFunctionReg.m by yourself)
[2] ex3_report – Directly give the answers of first three questions:
(1) Gradient at initial theta (zeros) for ex3data1
(2) Minimum cost at theta found by fminunc for ex3data1

(3) Predict an admission probability of a student with scores 70 and 50

(4) Train Accuracy for ex3data1 and ex3data2

(5) Find the best lambda which can get highest train accuracy for ex3data2(optional)

Throughout the exercise, you will be using the scripts `ex3.m` and `ex3reg.m`. This script set up the dataset for the problems and make calls to functions that you will write. You are required to modify `ex3.m` and some other functions, by following the instructions in this assignment.

Where to get help

The exercises in this course use Octave[1] or MATLAB, a high-level programming language well-suited for numerical computations.

At the Octave/MATLAB command line, typing help followed by a function name displays documentation for a built-in function. For example, help plot will bring up help information for plotting. Further documentation for Octave functions can be found at the Octave documentation pages. MAT- LAB documenttation can be found at the MATLAB documentation pages.

Do not look at any source code written by others or share your source code with others.

---

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 `ex3.m` will guide you through the exercise.to help you step through this exercise.

[1] Octave is a free alternative to MATLAB. For the programming exercises, you are free to use either Octave or MATLAB.

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 ex3.m, the code will load the data and display it on a 2-dimensional plot by calling the function plotData.

It displays a figure like Figure1, where the axes are the two exam scores, and the positive and negative examples are shown with different markers.

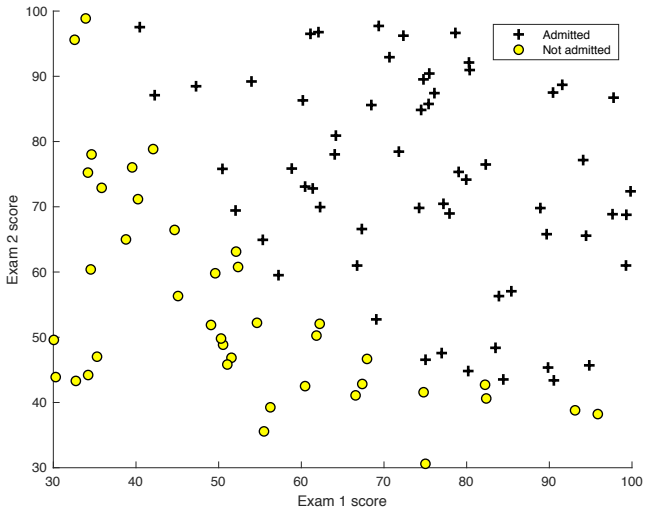

Figure 1: Scatter plot of training data

1.2 Implementation

1.2.1 Warmup exercise: 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 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.

1.2.2 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:

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

and the gradient of the cost is a vector of the same length as θ where the j^{th} element (for j = 0,1,...,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 $h_\theta(x)$.

1.2.3 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(θ) 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`, we already have code written to call fminunc with the correct arguments.

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

%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, `ex3.m` will call your costFunction function using the optimal parameters of θ.

This final θ value will then be used to plot the decision boundary on the training data, resulting in a figure similar to Figure2. We also encourage you to look at the code in `plotDecisionBoundary.m` to see how to plot such a boundary using the θ values.

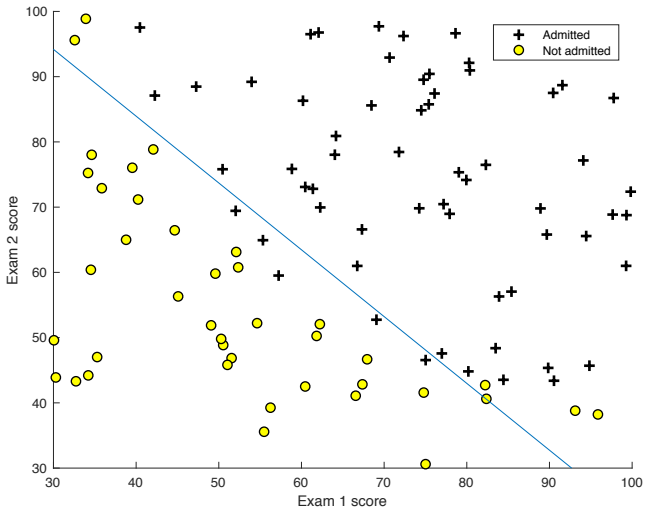

Figure 2: Training data with decision boundary

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 70 and an Exam 2 score of 50.

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.

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, `ex3reg.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 Figure3, 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.

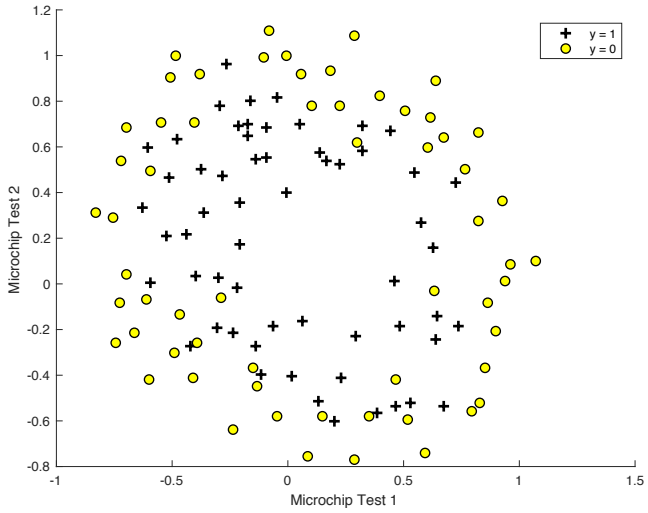

Figure 3: Plot of training data

Figure3 shows that our dataset cannot be separated into positive and negative examples by a straight-line through the plot. Therefore, a straight-forward 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 x1 and x2 up to the sixth power.

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.

$$\text{mapFeature}(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \\ x_1^3 \\ \vdots \\ x_1 x_2^5 \\ x_2^6 \end{bmatrix}$$

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 θ0. In Octave/MAT- LAB, 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:

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

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

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 `ex3reg.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 pro- vided the function plotDecisionBoundary.m which plots the (non-linear) decision boundary that separates the positive and negative examples. In plotDecision-Boundary.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 ex3reg.m will plot a decision boundary.

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 over- fitting.

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 4). 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.

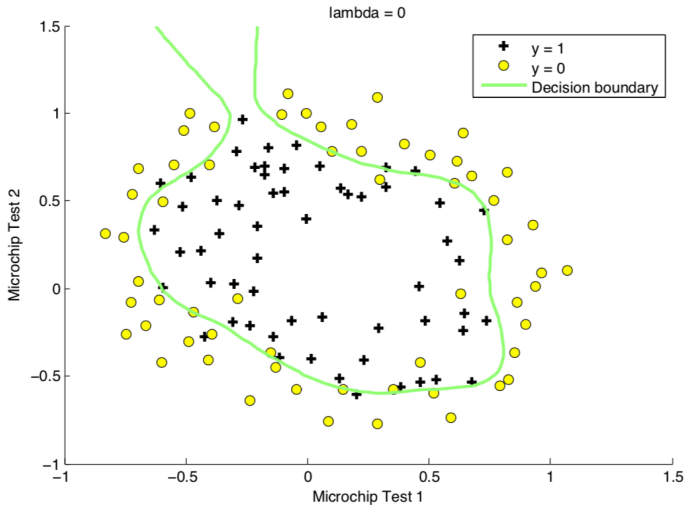

Figure 4: No regularization (Overfitting) (λ = 0)

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.

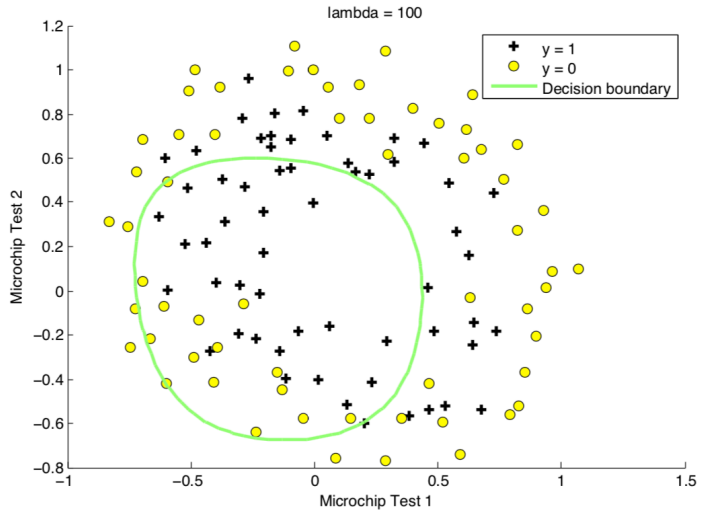

Figure 5: Too much regularization (Underfitting) ($\lambda = 100$)

Submission and Grading

After completing various parts of the assignment, be sure to submit all the files needed to blackboard. The following is a breakdown of how each part of this exercise is scored.

Part	Related code file	Points
Gradient at initial theta (zeros) for ex3data1	costFunction.m	20 points
Minimum cost at theta found by fminunc	costFunction.m	30 points
Predict an admission probability of a student with scores 70 and 50	ex3.m	10 points
Train Accuracy for ex3data1	predict.m	10 points
Train Accuracy for ex3data2	costFunctionReg.m	30 points
Total Points		100 points

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