Machine Learning	Programming Assignment 2
Comp540 Spring 2015	**due 6 February 2015 at 8 pm**

Introduction

In this assignment, you will implement logistic regression and regularized logistic regression and use it to analyze two different data sets. To get started, please download the code base `pa2.zip` from Owlspace. When you unzip the archive, you will see the following files.

Name	Edit?	Read?	Description
sigmoid.m	Yes	Yes	The sigmoid funcion
costFunction.m	Yes	Yes	Logistic regression cost function
predict.m	Yes	Yes	Logistic regression prediction function
costFunctionReg.m	Yes	Yes	Regularized logistic regression cost function
stdFeatures.m	Yes	Yes	Standardize features for spam data classification
logTransformFeatures.m	Yes	yes	log transform features for spam data classification
binarizeFeatures.m	Yes	Yes	threshold features for spam data classification
selectLambdaCrossval.m	Yes	Yes	select lambda by 10-fold crossvalidation
ex2.m	No	Yes	Matlab script that will run your functions for Part 1
ex2_reg.m	No	Yes	Matlab script that will run your functions with regularization for Part 2
ex2_spam.m	No	Yes	Matlab script that will run your functions for spam data classification (Part 3)
mapFeature.m	No	Yes	Function to generate polynomial features
plotData.m	No	Yes	Function to plot 2D classification data
plotDecisionBoundary.m	No	Yes	Function to plot classifier's decision boundary
ex2data1.txt	No	No	Dataset for the first part of the assignment
ex2data2.txt	No	No	Dataset for the second part of the assignment
spamData.mat	No	No	Dataset for the third part of the assignment
pa2_2015.pdf	No	Yes	this document

Problem 1: Logistic regression (15 points)

In this problem, you will implement logistic regression to predict whether a student gets admitted into a university. Suppose you are an admissions officer and you want to determine an applicant's chance of admission based on the scores on two exams. You have historical data from previous applicants consisting of their scores on the two exams and the admission decision made. Your task is to build a classifier that estimates the probability of admission based on the two scores.

Visualizing the dataset

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`. Run the script `ex2.m` and it will plot the training data as shown in Figure 1, where the axes are the two exam scores, and the positive and negative examples are shown with different markers.

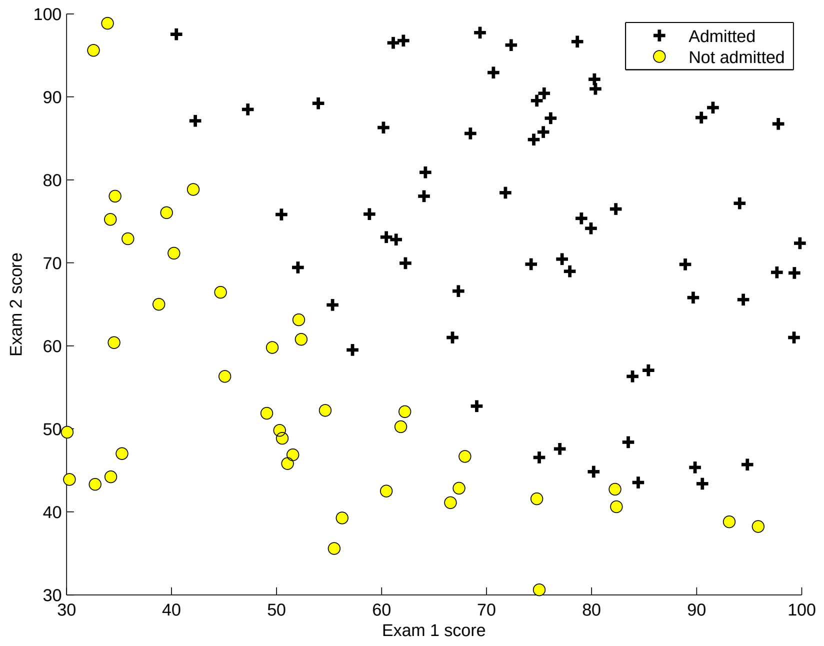

Figure 1: The training data

Implementing logistic regression : the sigmoid function (5 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 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 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.

Cost function and gradient of logistic regression (5 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. The cost function in logistic regression is

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

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

Learning parameters using fminunc

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 θ. Use `fminunc` to find the best parameters for the logistic regression cost function, given a fixed dataset (of `X` and `y` values). You will pass `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.

```
% 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`.

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.

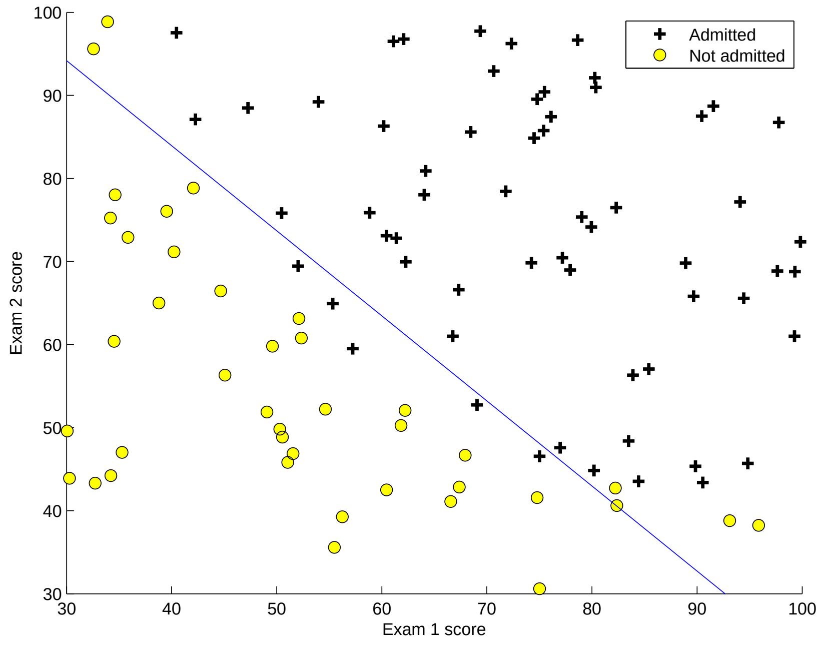

Figure 2: The decision boundary

Evaluating logistic regression (5 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 about 0.774. 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. You should expect to see 89% accuracy on the training data.

Problem 2: Regularized logistic regression (15 points)

In this part of the exercise, you will implement regularized logistic regression to predict whether microchips from a fabrication plant pass 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.

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 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.

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.

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.

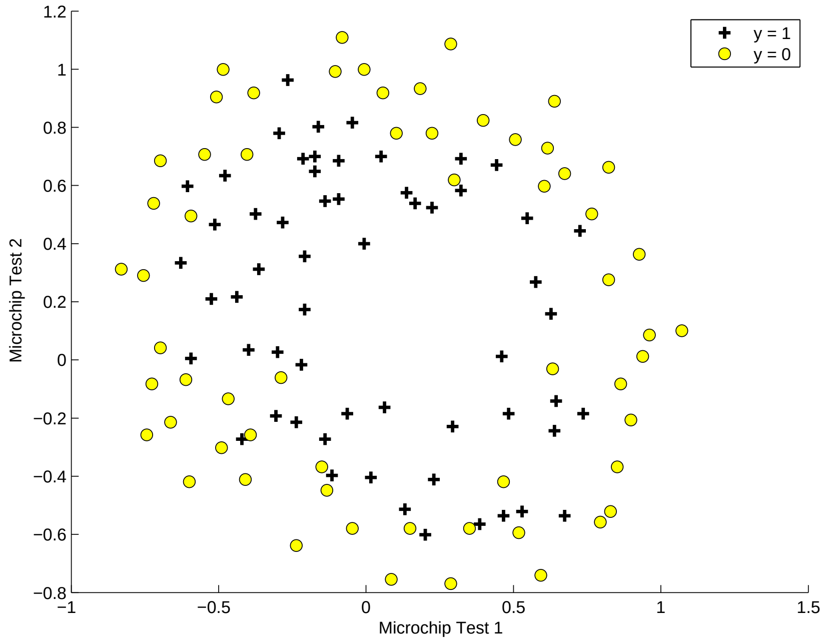

Figure 3: Plot of training data

Cost function and gradient (10 points)

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. The regularized cost function is:

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

Note that we do not regularize θ_0. The gradient of the cost function is a vector where the j^{th} element is defined as follows:

$$\begin{aligned}
\frac{\partial J(\theta)}{\partial \theta_0} &= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \quad \text{for } j = 0 \\
\frac{\partial J(\theta)}{\partial \theta_j} &= \left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{2m}\theta_j \quad \text{for } j \geq 1
\end{aligned}$$

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`.

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 classifiers predictions on an evenly spaced grid and then draw a a contour plot of where the predictions change from y = 0 to y = 1. After learning the parameters θ, the next step in `ex2_reg.m` will plot a decision boundary similar to Figure 4.

Varying λ (5 points)

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 (say 0), you should find that the classifier gets almost every training example correct, but draws a very complicated boundary, thus overfitting the data. With a larger λ, you should get a simpler decision boundary which still separates the positives and negatives fairly well. However, if λ is set to too high a value (say 100), you will not get a good fit and the decision boundary will not follow the data so well, thus underfitting the data. Show plots of the decision boundary for two lambdas showing overfitting and underfitting on this data set.

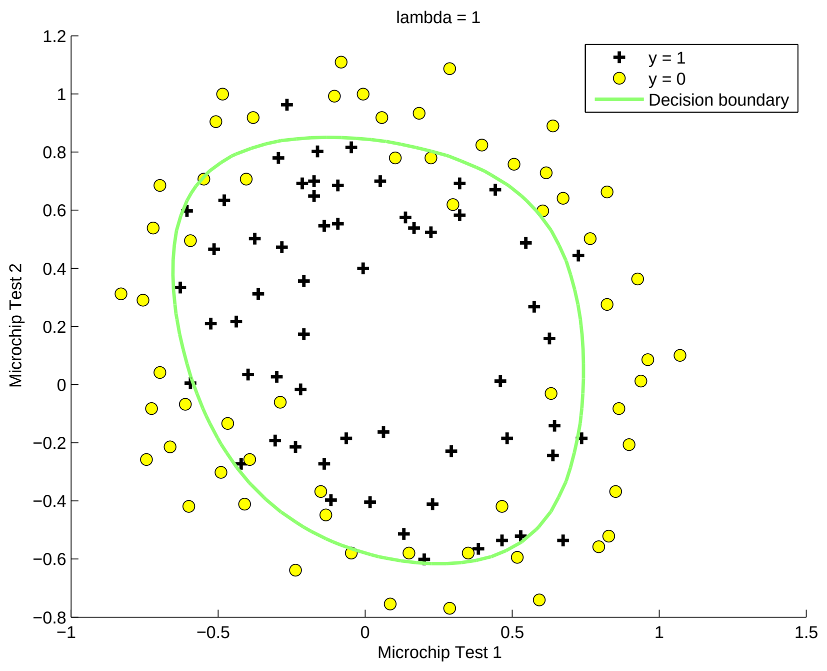

Figure 4: Training data with decision boundary for `lambda` = 1

Problem 3: Logistic regression for spam classification (20 points) [8]

(Source: Kevin Murphy) Consider the email spam data set developed by Hastie et. al. It has 4601 email messages, from which 57 features have been extracted. This data is in `spamData.mat` which has a training set of size 3065 and a test set of size 1536. The features are as follows:

- 48 features in [0,100], giving the percentage of words in a given email which match a given word on the list. The list contains words such as "business", "free", "george", etc. The data was collected by George Forman, so his name occurs quite a lot.
- 6 features in [0,100], giving the percentage of characters in the email that match a given character on the list. The characters are ;, (, [, !, $, #.
- Feature 55: the average length of an uninterrupted sequence of capital letters (max is 40.3, min is 4.9).
- Feature 56: the length of the longest uninterrupted sequence of capital letters (max is 45.0, mean is 52.6).
- Feature 57: the sum of the lengths of uninterrupted sequence of capital letters (max is 25.6, mean is 282.2).

Feature transformation (6 points)

Scaling features is important in logistic regression. Here you will implement three methods for transforming features in the spam data set: `stdFeatures`, `logTransformFeatures` and `binarizeFeatures`. There are *.m* files with these names in our directory and you will complete the code in them. Here are the descriptions of the transformations.

- Standardize features: transform each column of the data set by subtracting the mean of the column and dividing by the standard deviation. Thus each column has a mean of zero and unit variance.
- Log transform features: replace every $x_j^{(i)}$ by $log(1 + x_j^{(i)})$.
- Binarize features: replace every $x_j^{(i)}$ by 1 if $x_j^{(i)} > 0$ or 0 otherwise.

Fitting regularized logistic regression models (14 points)

For each representation of the features, we will fit regularized logistic regression models. Your task is to complete the function `select_lambda_crossval` to select the best regularization parameter λ by 10-fold cross-validation on the training data. This function takes a training set `X` and `y` and sweeps a range of λ's from `lambda_low` to `lambda_high` in steps of `lambda_step`. Default values for these parameters are in `ex2_spam`. For each λ, divide the training data into ten equal portions using the Matlab function `crossvalind`. Train a logistic regression model on nine of those parts and test its accuracy on the left out portion. The accuracy of a model trained with that λ is the average of the ten test errors you obtain. Do this for every λ in the swept range and return the lambda that yields the highest accuracy .

`ex2_spam` will then build the regularized model with the best lambda you calculate and then determine the training and test set accuracies of the model. With features generated by `stdFeatures` you should see an accuracy of 91.8% on training data and 91.3% on test data. With `logTransformFeatures` you should get training data accuracy of 94.8% and test set accuracy of 93.9%. With `binarizeFeatures` you should see training set accuracy of 93.5%and a test set accuracy of about 92.6%.

What to turn in

Please zip up all the files in the archive (including files that you did not modify) and submit it as `pa2_netid`.zip on Owlspace before the deadline. Include a PDF file in the archive that presents your plots and your discussion of results from the problems above.

Acknowledgment

Problems 1 and 2 are adapted from Andrew Ng's exercise on logistic regression.